빌드 배포 문서

- IntelliJ IDEA Community Edition(2021.3.1 ver)
- MobaXterm Home Edition
- Docker
- mysql workbenck CE 8.0
- Gradle 6.7
- NodeJS 16.13.0
- Openvidu 2.20.0

1. EC2에 mysql 설치
    a. `sudo apt update`

    업데이트가 필요한 내역들을 리스트업 해줌

    b. `sudo apt install mysql-server`

    mysql 설치

    c. `sudo mysql -u root -p`

    root유저로 mysql 접속하기

    d. 현재 비밀번호 입력해서 mysql 접속

    e. `show databases;`

    f. `use mysql;`

    mysql 데이터베이스에 접속

2. mysql에 user 생성
    a. `create user '유저명'@'%'`

    모든 사람이 접속가능한 user 생성

    `create user '유저명'@'localhost'`

    `%`를 localhost로 작성하면 나만 접속이 가능해진다.

    b. `create user '유저명'@'%' IDENTIFIED BY '비밀번호';`

    주소에 비밀번호 부여

    c. `grant all privileges on *.* to '유저명'@'%';`

    모든 권한 부여

    d. `flush privileges;`

    mysql에 변경사항 적용

    e. `cd /etc/mysql/mysql.conf.d`

    `mysql.conf.d` 폴더의 `mysqld.cnf` 파일의 세팅을 변경하기 위해 해당 경로로 이동

f. `sudo nano mysqld.cnf`

세팅 내용을 변경하기 위해 루트권한으로 접속

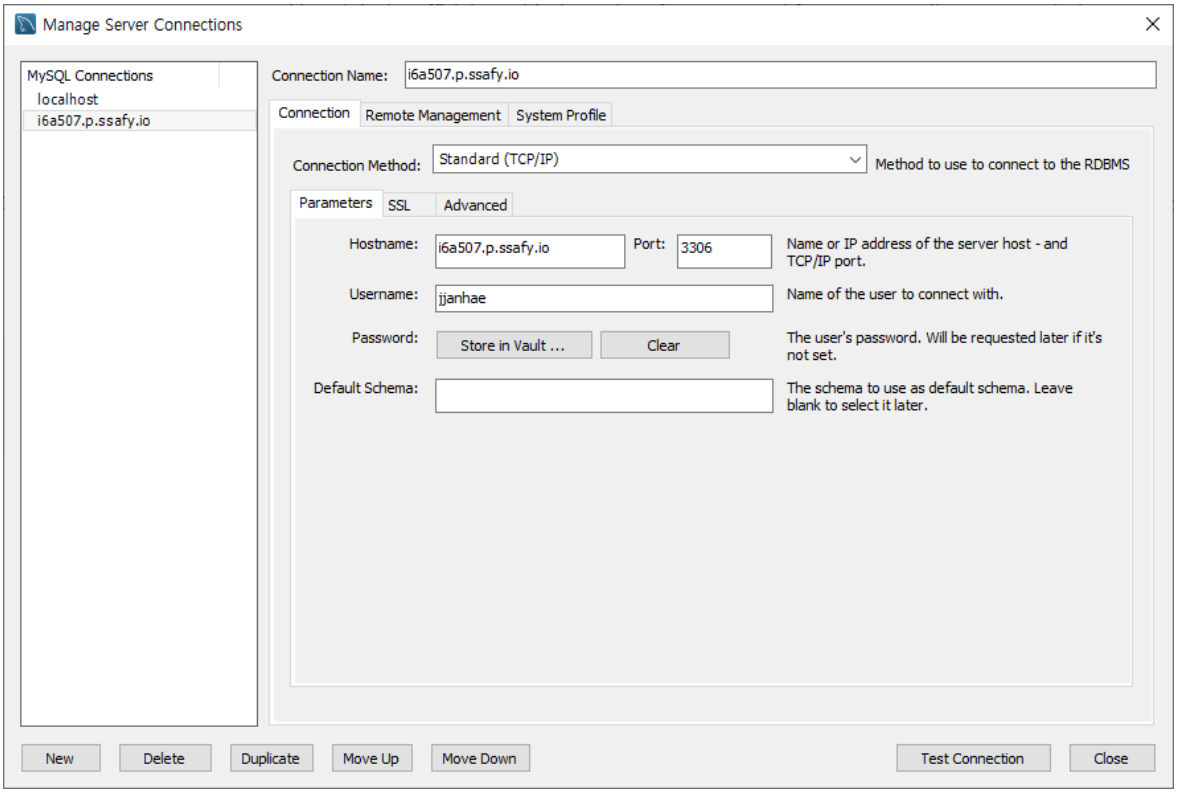
g. `bind-address` `0.0.0.0`으로 변경

모든 사람들에게 접속권한이 부여됨

h. `sudo service mysql restart`

mysql 재시작하여 변경내용 적용

i. mysql workbench 접속



Hostname: i6a507.p.ssafy.io

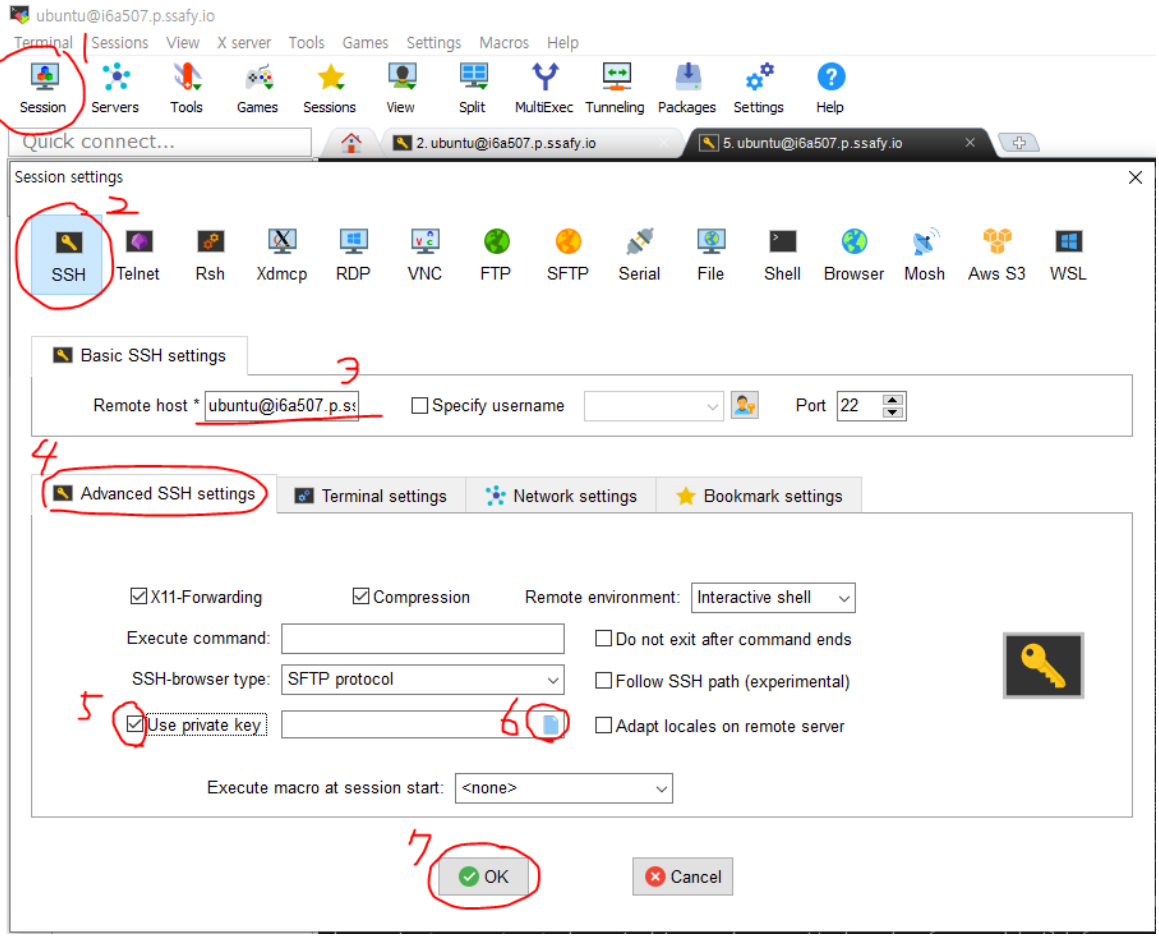
Port: 3306

Username: jjanhae

Password: password

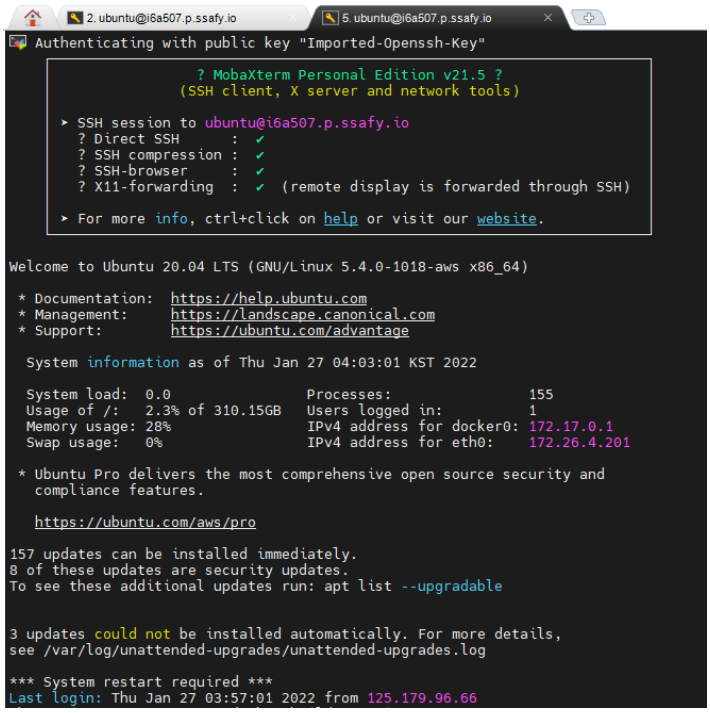
위와 동일하게 입력 후 Test Connection

3. EC2 접속
   a. Session 탭 클릭하여 접속하기



1. **Session** : 누른다. 그러면 Session settings 창이 뜸
2. **SSH** : 우리는 ec2서버로 원격으로 접속해서 빌드배포 해야하니깐 SSH를 누른다.
3. **Remot host** : 접속할 서버 주소. ubuntu로 접속해야 해서 앞에 ec2 주소 앞에 `ubuntu@` 를 붙여준다. username설정은 안해줘도 되더라. 그러므로 나도 굳이 하지 않는다.
4. **Advanced SSH settings** : ssh로 접속해야 하니까 setting해야함. 여기서 제공받은 PEM키를 입력
5. **Use private key** : 체크한다.
6. PEM키가 저장되어있는 경로 지정
7. **OK** : 접속

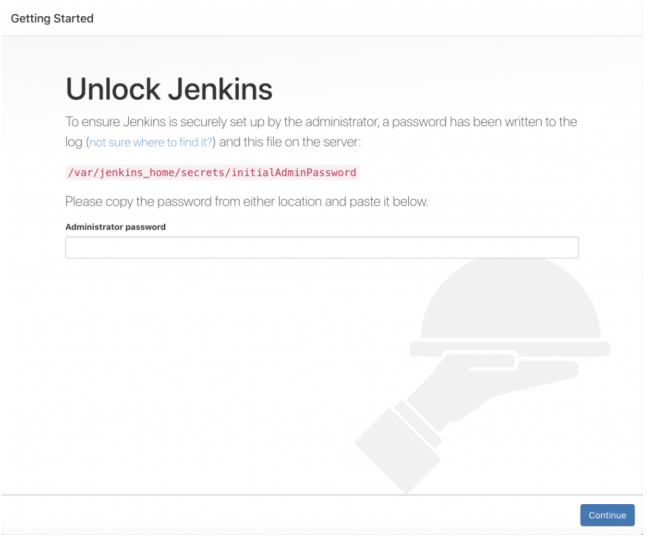
b. `ssh -i C:/경로/I6A507T.pem ubuntu@i6a507.p.ssafy.io` 입력



접속 성공 시 화면

4. Docker 위에 Jenkins 설치
   a. `curl -fsSL https://get.docker.com/ | sudo sh`
   b. `sudo usermod -aG docker $USER`

      현재 접속한 유저에게 도커 실행권한을 줌
   c. `docker run -d -p 8080:8080 -v /home/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -u root jenkins/jenkins:lts`
   d. Jenkins 초기 설정



창에서 지시하는 대로 비밀번호를 받아 적어야 한다.

이를 위해 docker 컨테이너 내부로 접속

e. `docker ps -a`

jenkins 컨테이너의 `CONTAINER ID` 확인

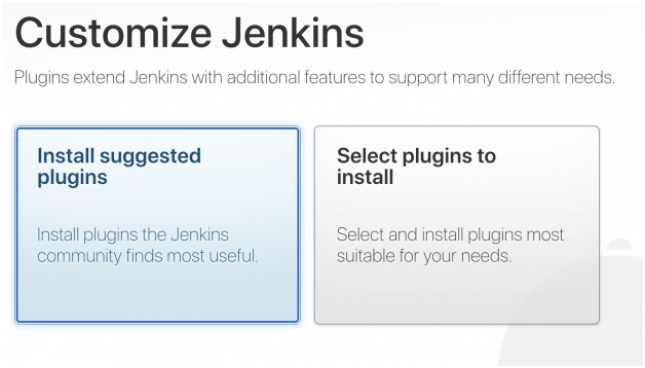
f. `docker exec -it (CONTAINER ID) /bin/bash`

docker는 `CONTAINER ID`의 첫 몇 자만 입력해도 인식한다.

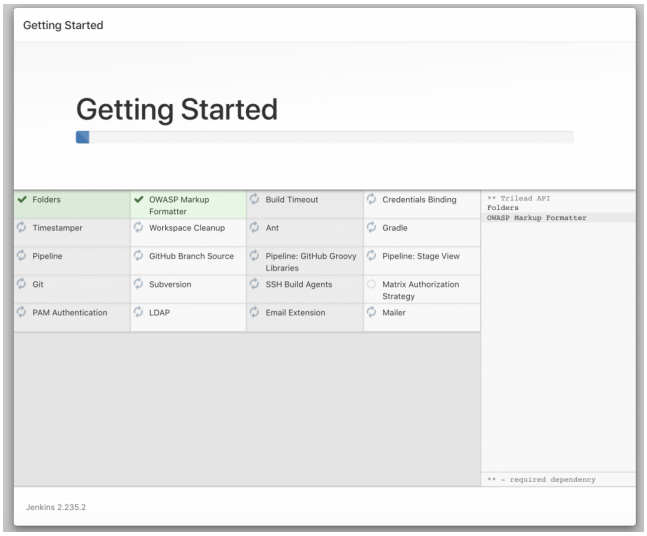
g. `root@(CONTAINER ID):/# cat /var/jenkins_home/secrets/initialAdminPassword`

초기 비밀번호가 출력됨

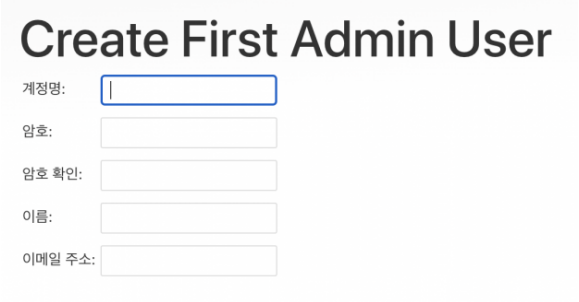
획득한 비밀번호를 Unlock Jenkins 창에 붙여넣고 Continue



Install suggested plugins 클릭



기다린다.

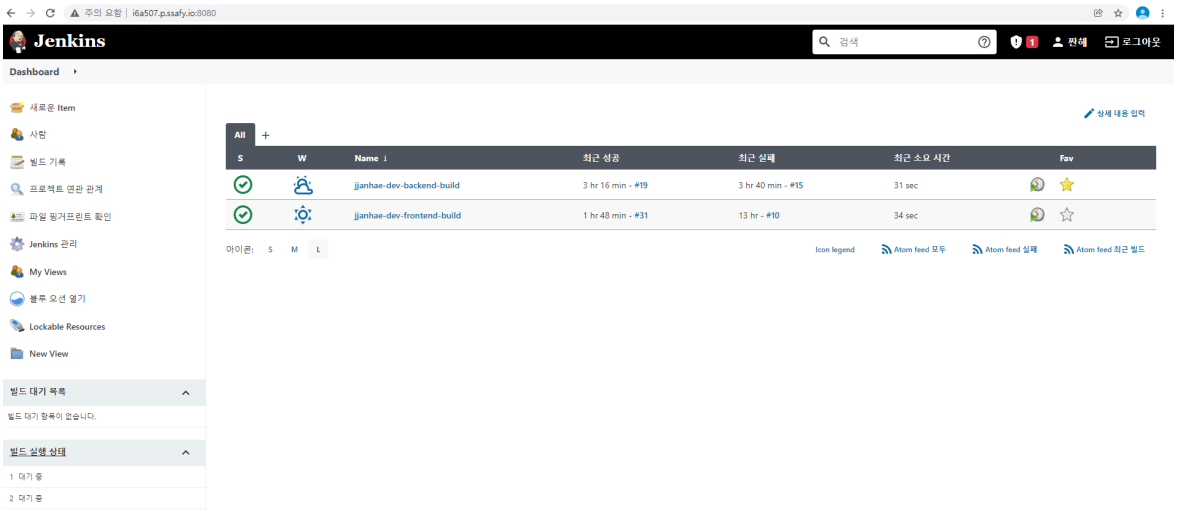


관리자 계정 생성

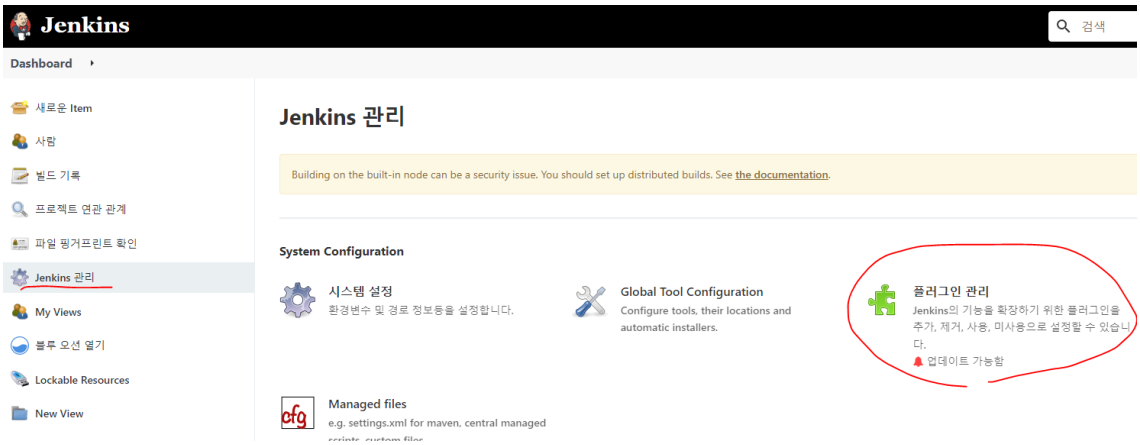
5. 인프라 구축
   a. 관리자 계정으로 로그인

      계정명 : jjanhae

      비밀번호 : ssafy6jjanhae!!



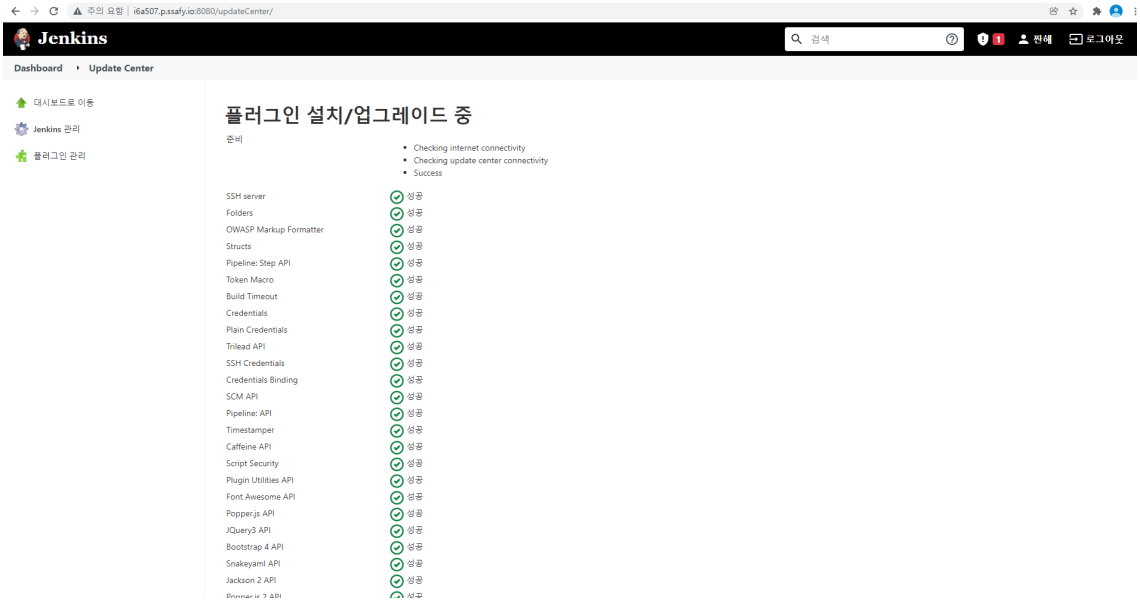
   b. Jenkins 환경설정
      a. 플러그인 설치

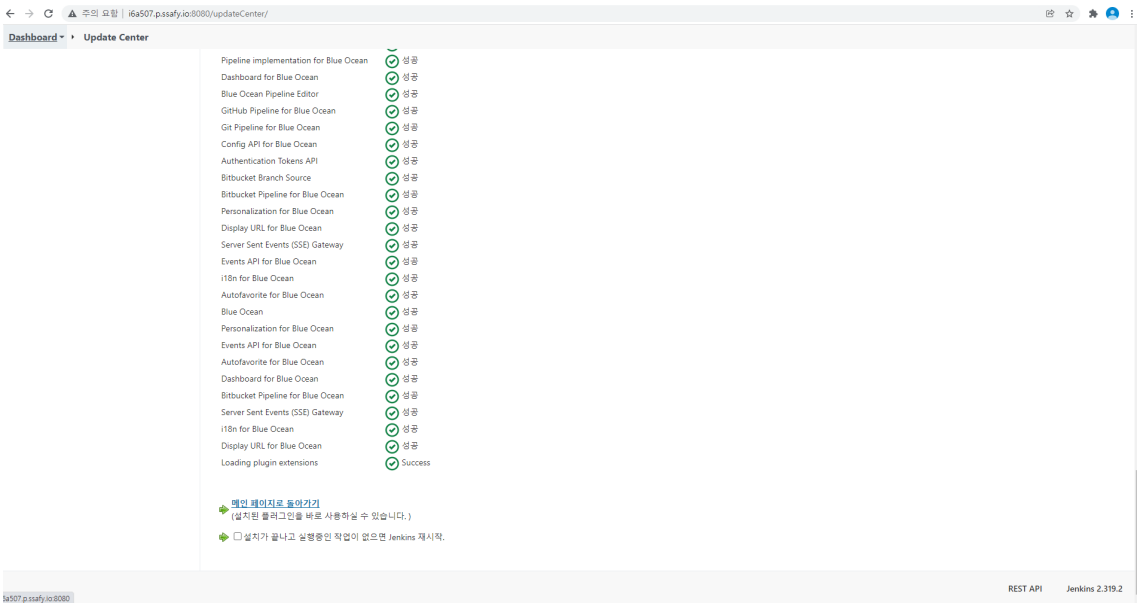


플러그인 설치부터 해야한다.

여기서 nodejs, gitlab, blue ocean 관련된것을 다 설치한다.

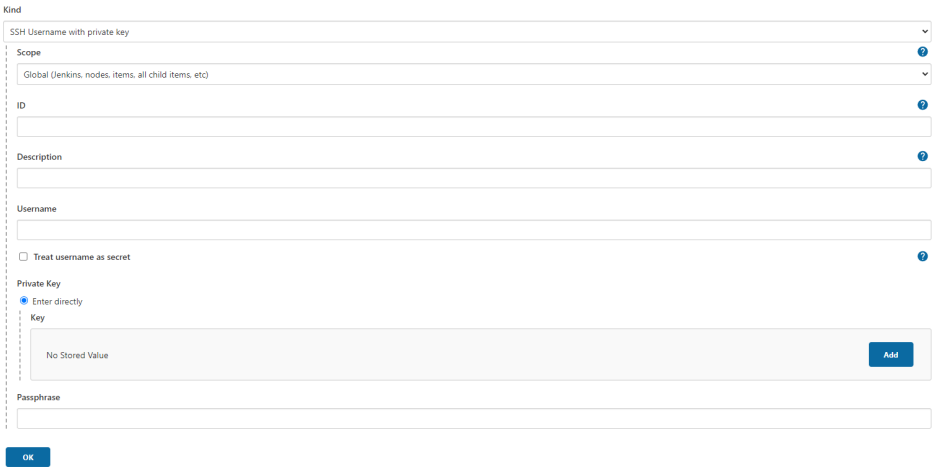
설치할거 검색해서 다 체크해놓고 Install without restart 클릭





엄청 많이 설치됨

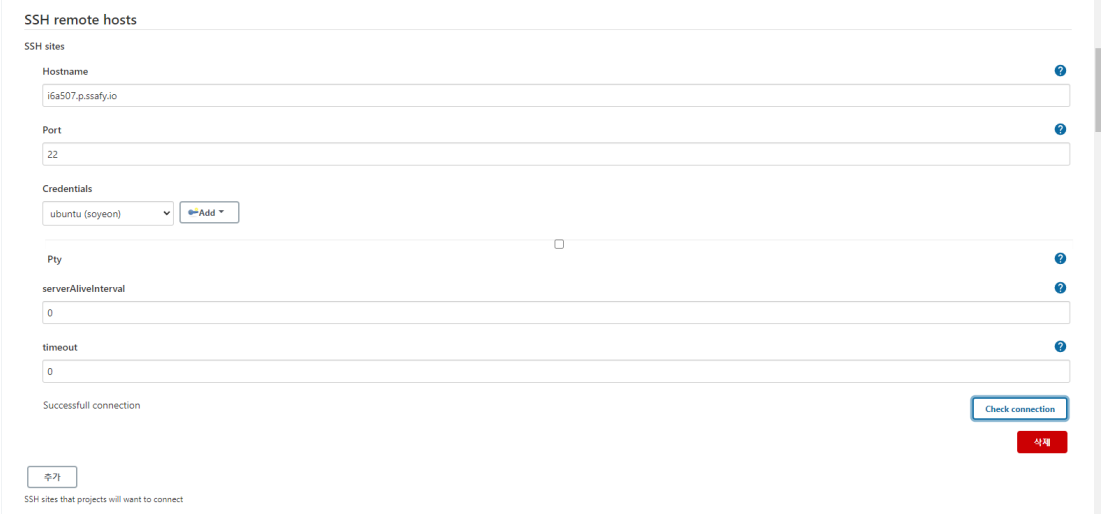
b. ssh 연결



1. **Kind** : SSH Username with private key 선택
2. **Scope** : Global
3. **ID** : 알아볼수 있고 고유한거 아무거나로 설정
4. **Description** : 안써도되거나 아무거나
5. **Username** : 이게 중요하다.
   결국 ubuntu@i6a507.p.ssafy.io 로 접근해야 하므로 Username으로 `ubuntu` 로 설정해줬어야함
6. **Private Key**
   a. Enter directly 체크
   b. 로컬에 저장된 PEM키를 IDE로 열어서 전체복사
   c. Add 버튼 눌러서 붙여넣기

7. **Save**

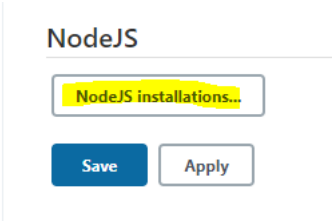
그리고 다시 `대쉬보드 > 환경설정 > SSH remote hosts` 로 이동



1. Credential key설정에 Username을 쓰지않고 Hostname `ubuntu@i6a507.p.ssafy.io` 로 접근
2. Credential key설정에 Username `ubuntu` 로 설정, Hostname `i6a507.p.ssafy.io` 로 접근

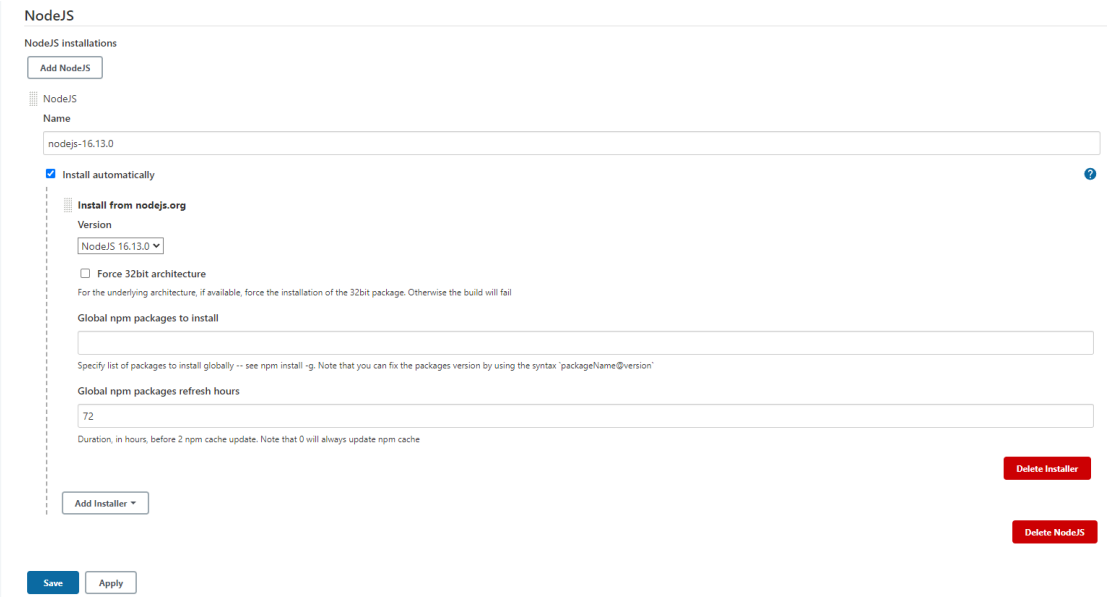
이 두가지 중 한 가지로 해야 한다.

c. 기타 환경설정

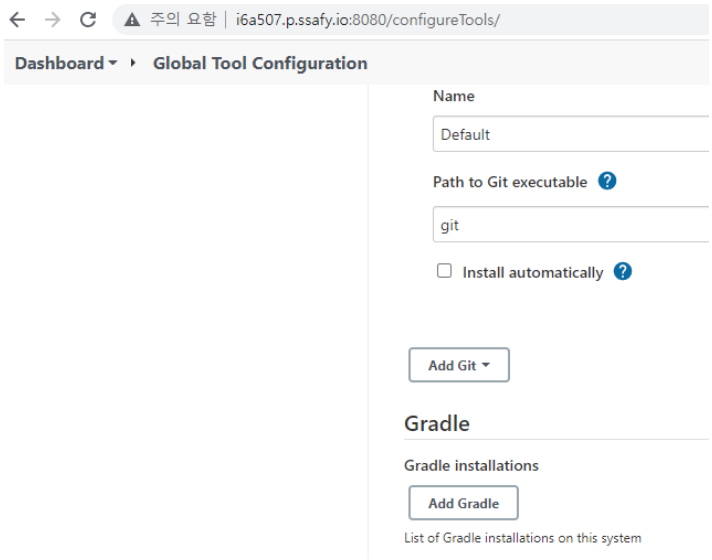


홈 > Jenkins 관리 > Global Tool Configuration > NodeJS

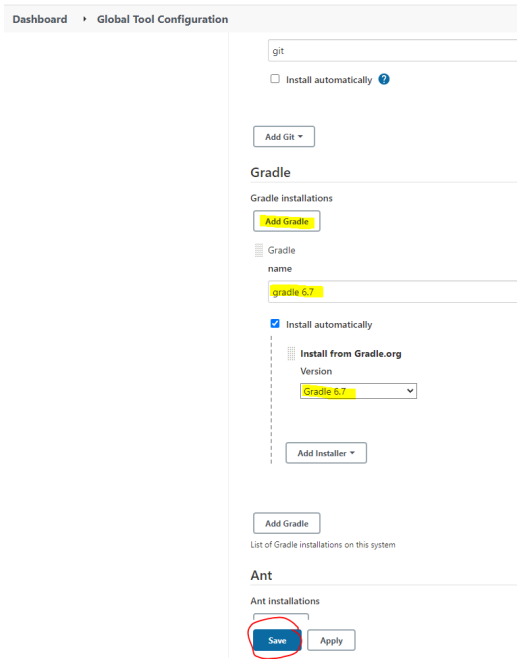
NodeJS installations... 클릭



NodeJS 16.13.0 설정해주면 빌드환경에서 Provide Node & npm .. 눌렀을때 NodeJS가 옵션으로 뜨게 된다.



global tool configuration 에서 gradle설정

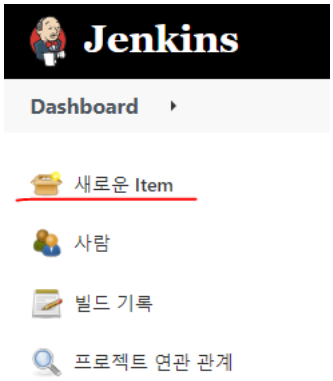


순서대로 입력하고 save

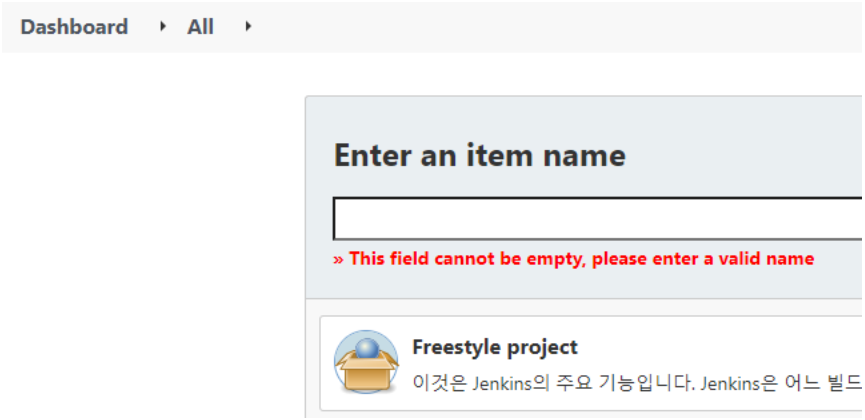
다시 메인으로 가서, 프로젝트 '구성' 클릭

이제 환경설정은 끝났고 job 구성이 남았다.

c. backend 자동빌드배포환경 구축



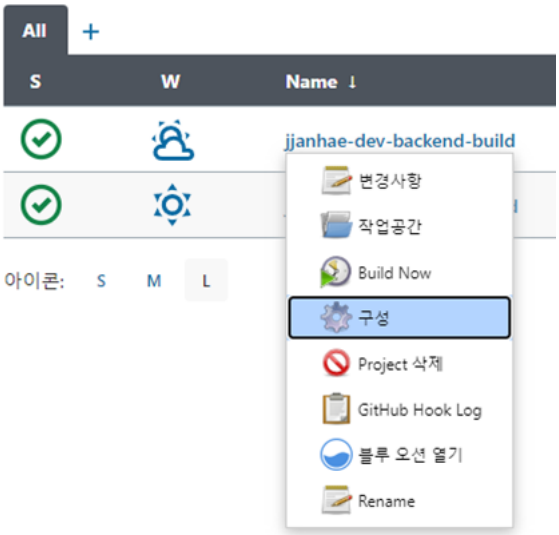
클릭



이름 정하고

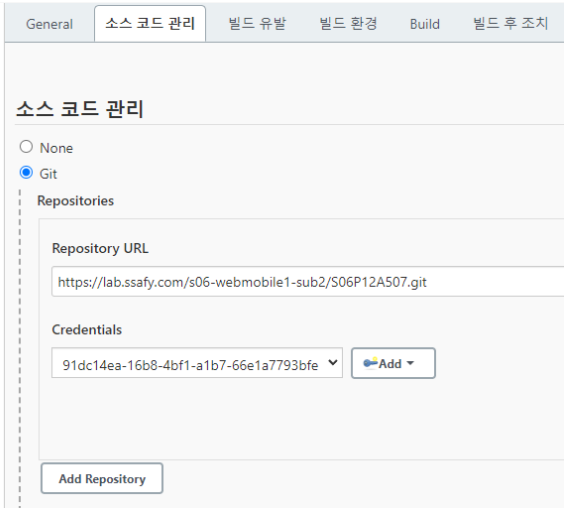
Freestyle project 클릭

OK

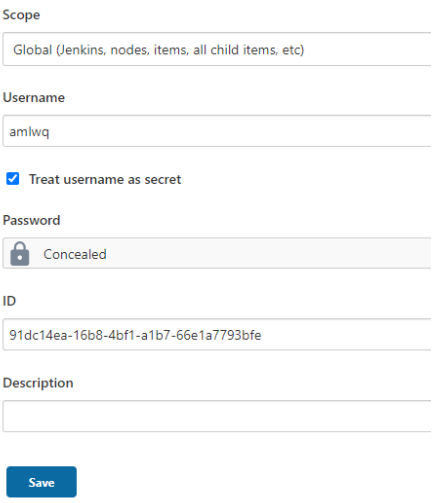


Name 옆에 마우스 갖다대면 ▼ 표시가 뜨는데, 그거누르고 구성 클릭

d. jenkins & gitlab 연동

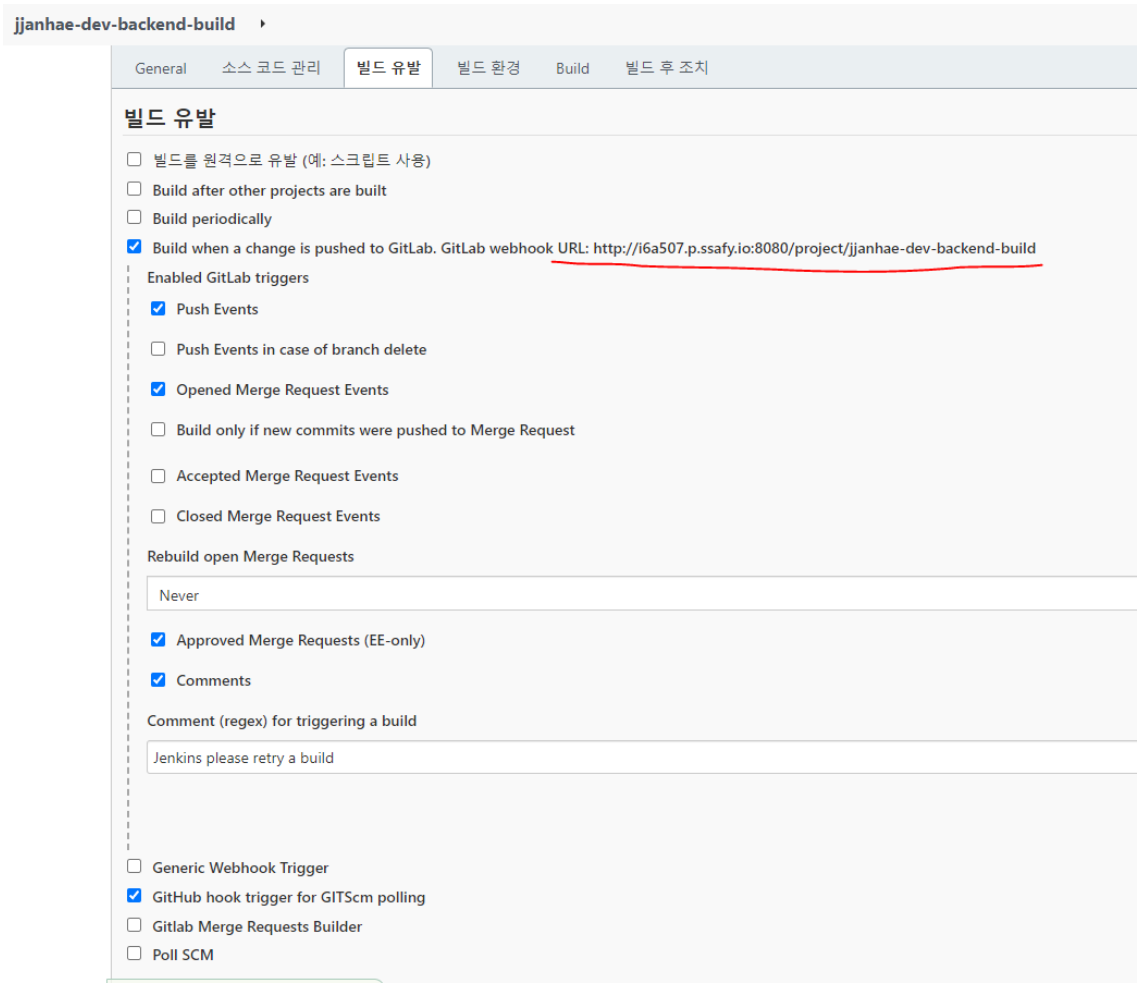


1. **Git** 체크
2. **Repository URL** : 연동할 gitlab의 https를 복붙
3. **Credentials** : 개인키 생성



- **Scope** : Global
- **Username** : gitlab 개인 ID
- **Password** : gitlab 개인 PW
- **ID** : 아무거나

e. webhook



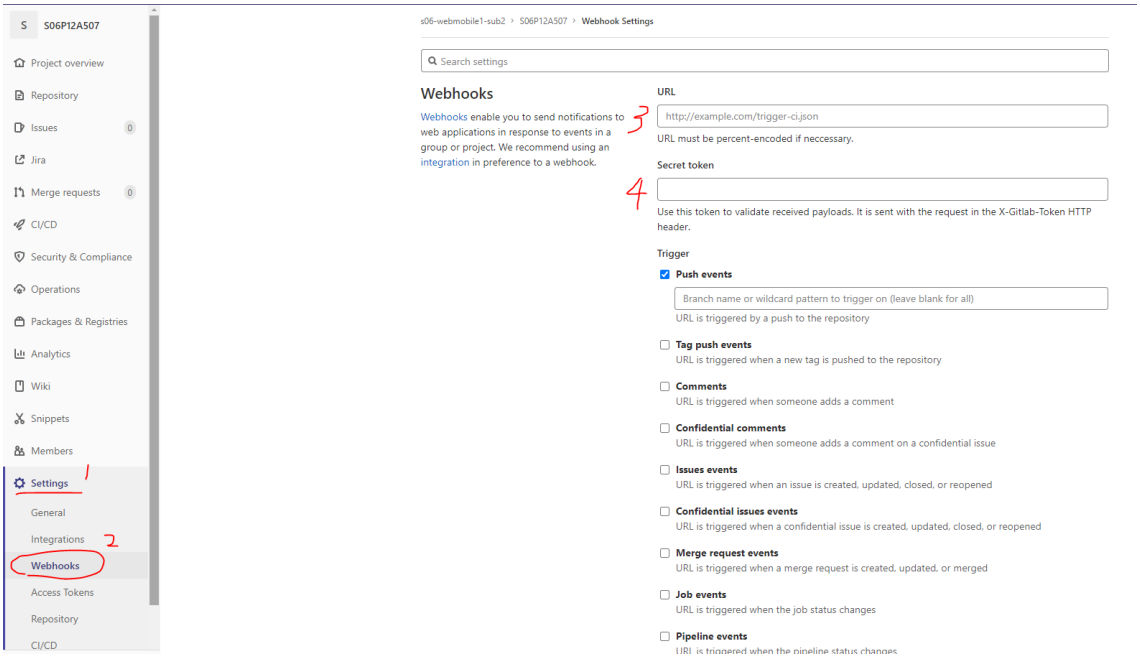
Build when a change is ... 체크

저 설정이 gitlab에서 push가 들어오면 자동으로 빌드해주는 옵션이다.

`고급` 버튼을 누르면 옵션들이 더 나타나게 되는데, 여기서 `Secret token` 옵션의 Generate를 클릭하여 토큰을 발급한다.

이 발급된 토큰을 gitlab에 입력해주어야 한다.

빨간밑줄에 해당하는 URL과 저 토큰을 복사 후 gitlab 을 연다.

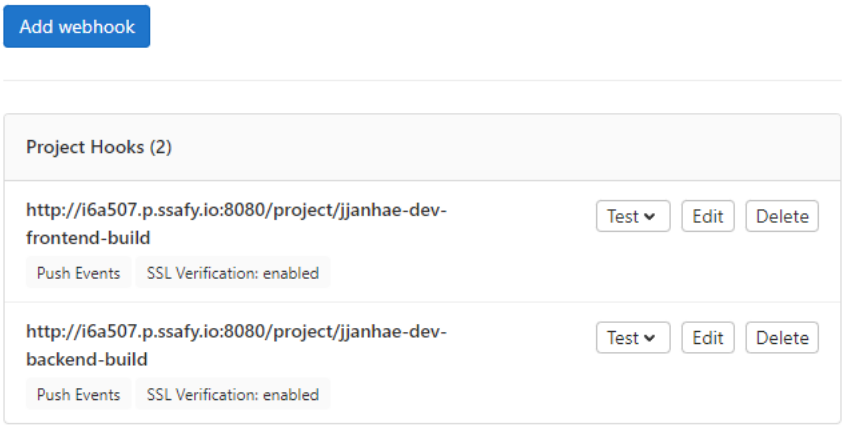


Settings > Wehbooks 로 접근

아까 복사했던 URL과 토큰을 3,4번에 차례대로 붙여넣기한다.

그리고 Push events로 젠킨스 빌드를 발생시킬 브랜치를 넣는다.

안넣어주면 모든 브랜치에 푸쉬할때마다 자동 빌드된다.



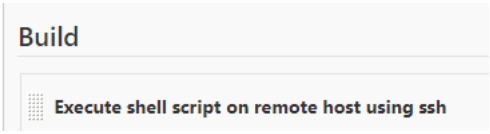
아래로 내려서 `Add webhook` 버튼을 누르면, 아래의 Project Hooks 에 한줄 추가된다.

여기까지 했으면 gitlab에 push 시 자동 빌드까진 된다.

그런데 우리는 빌드 후 ec2 서버에 배포까지 자동설정 해주어야 한다.

자동 배포 설정

그러기 위해서는 jenkins에서 ssh로 원격조종 해줘야 한다.



Excute shell script on remote host using ssh 선택

command로 ssh에서 동작할 명령어을 넣어줘야 한다.

`deploy.sh` 파일을 실행시켜 배포해야 한다.

```
ubuntu@ip-172-26-4-201:/home/jenkins/workspace$ cd jjanhae-dev-backend-build
ubuntu@ip-172-26-4-201:/home/jenkins/workspace/jjanhae-dev-backend-build$ cd backend
ubuntu@ip-172-26-4-201:/home/jenkins/workspace/jjanhae-dev-backend-build/backend$ ls
README.md  build.gradle  deploy.sh  dockerfile  gradle  gradlew  gradlew.bat  jjanhae.sql  settings.gradle  src
ubuntu@ip-172-26-4-201:/home/jenkins/workspace/jjanhae-dev-backend-build/backend$ sudo vi deploy.sh
```

모바엑스텀에서 `/home/jenkins/workspace/jjanhae-dev-backend/backend` 디렉토리로 이동

jenkins ssh shell에서 실행시킬 deploy.sh 파일을 생성해야 함.

`sudo vi deploy.sh` 실행

앞에 `sudo` 명령어를 붙여줘야 모든 실행 권한을 가진다. 안해주면 권한없다고 파일 쓰기 및 수정 못하게됨.

`i` 를 눌러 수정모드로 바꾸고 아래의 코드 입력

```
# 빌드를 빠르게 함, 로그 출력
DOCKER_BUILDKIT=1 docker build --progress=plain -t jjanhae .

# jjanhae 컨테이너 삭제
docker rm jjanhae -f

# jjanhae 이미지로 jjanhae 컨테이너를 생성, 내외부 포트 8081로 하고 실행
docker run -d -p 8081:8081 --name jjanhae jjanhae

# 동작하지 않는 이미지 정리
docker image prune -f
```

`ESC` 를 눌러 수정모드 종료

`:wq` 를 입력해 저장하고 빠져나온다.

이제 dockerfile 만들 차례

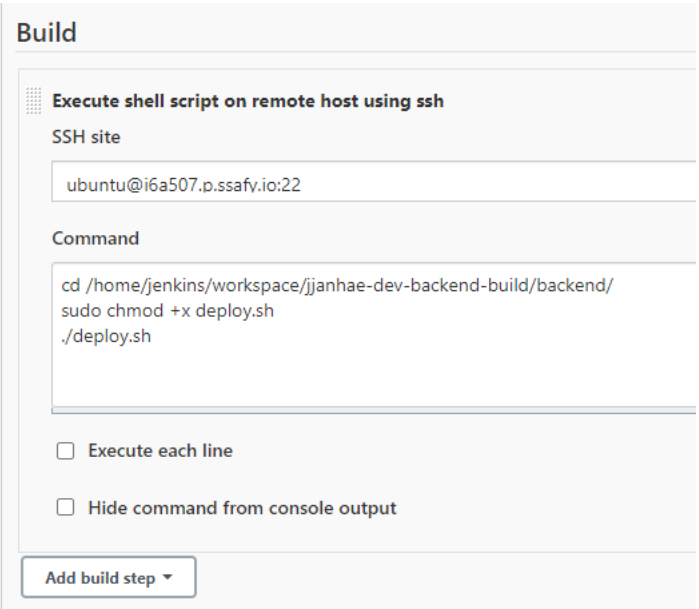
마찬가지로 `sudo vi dockerfile` 입력

```
FROM adoptopenjdk/openjdk8:x86_64-alpine-jdk8u292-b10 as builder
WORKDIR /app
COPY build.gradle settings.gradle gradlew ./
COPY gradle ./gradle
RUN \
chmod +x ./gradlew \
&& ./gradlew build \
|| return 0
COPY src ./src
```

```
RUN ./gradlew bootJar
FROM adoptopenjdk/openjdk8:x86_64-alpine-jre8u292-b10
WORKDIR /app
COPY --from=builder /app/build/libs/*.jar ./app.jar
EXPOSE 8081
ENTRYPOINT ["java", "-jar", "app.jar"]
CMD ["--datasource.username=jjanhae", "--datasource.password=password"]
```

저장 후 빠져나옴

다시 젠킨스로 이동



커맨드에 다음과 같이 작성.

`cd /home/jenkins/workspace/jjanhae-dev-backend-build/backend/`

아까 모바엑스텀에서 작성한 파일이 있는 경로로 이동

`sudo chmod +x deploy.sh`

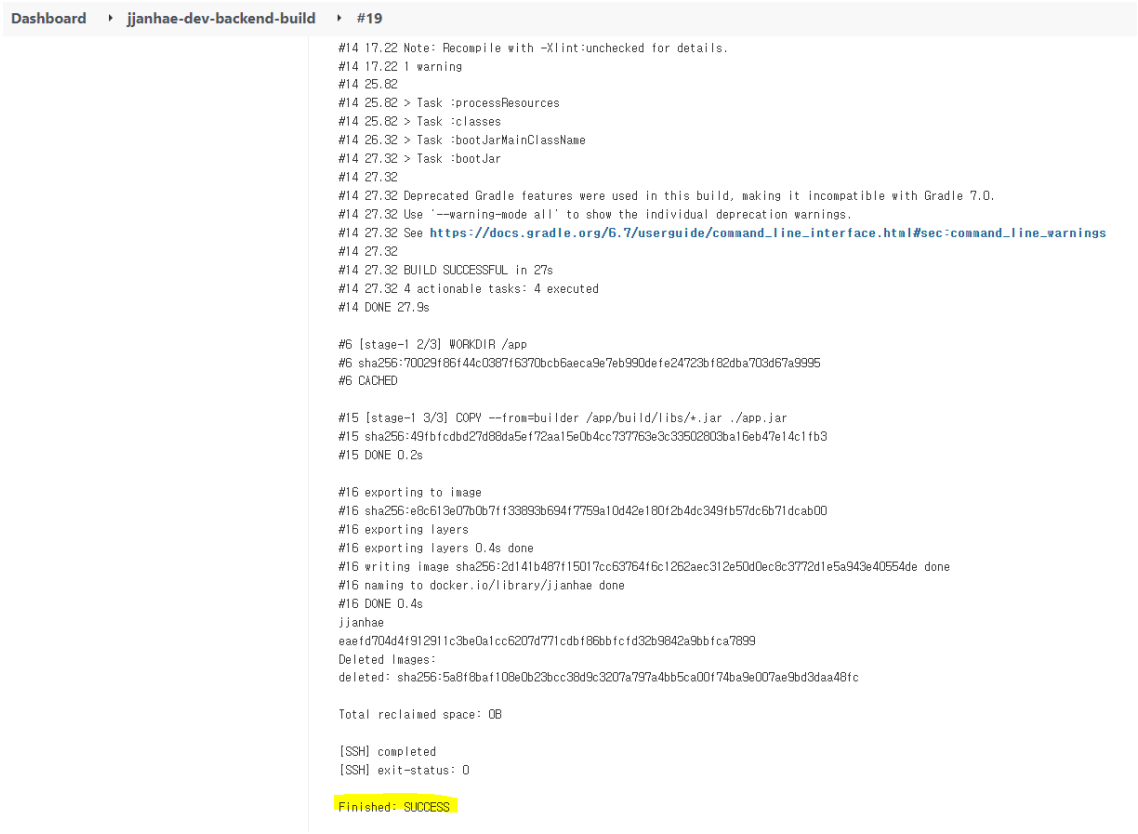
deploy.sh 파일을 실행시킬 권한 부여

안해줄 시 실행 권한없다고 거부당함

`./deploy.sh`

deploy.sh 실행

그리고 저장 후 Build Now로 빌드시킴



SUCCESS

backend 자동빌드배포환경 구축

앞서 백엔드 인프라를 구축할 때, 환경설정을 이미 다 마져주었으므로 프론트 인프라를 구축할 땐 생략해도 된다.

gitlab연동과 web hook만 똑같이 해주면 됨

nginx는 기본적으로 80포트를 사용하고, http로 요청할 경우 80포트는 생략된다.

https는 443

따라서 프론트는 nginx 위에 올리는게 맞다는 생각이 들었다.

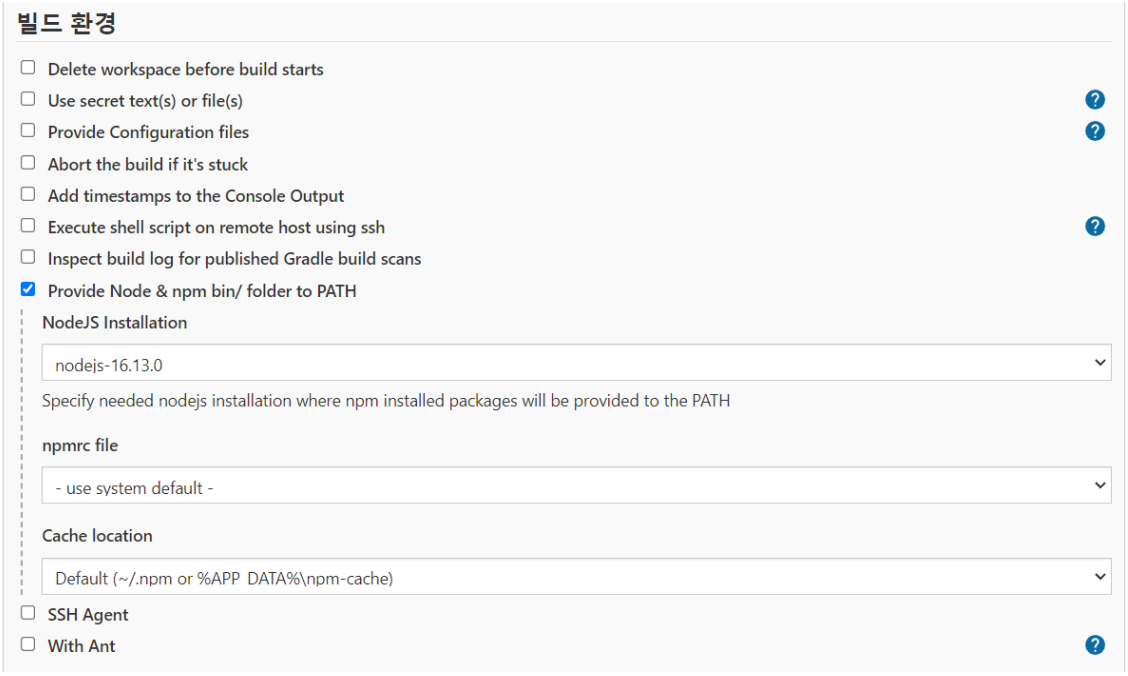
기본적으로 위 링크는 npm build된 것을 nginx에 복사하는 방식이다.

일단 작은 단계로 나눠서 진행한다.

1. jenkins에서 npm build하여 build폴더 생성하기
2. 생성된 build 폴더를 nginx에 복사하기

1. jenkins npm build

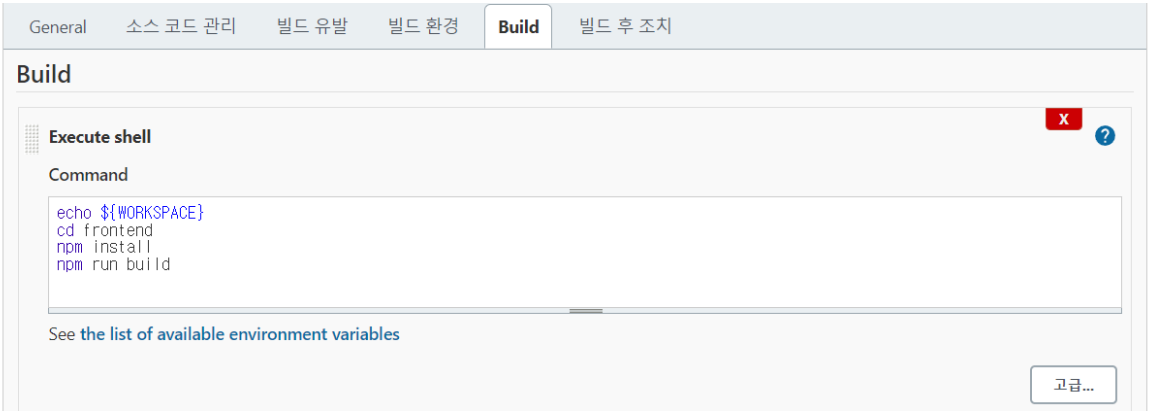
jjanhae-dev-frontend-build > 구성



Provide Node & npm 체크

맞는 NodeJS 버전으로 설정

만약 저 옵션이 없다면 npm plugin이 설치되어 있는지 확인



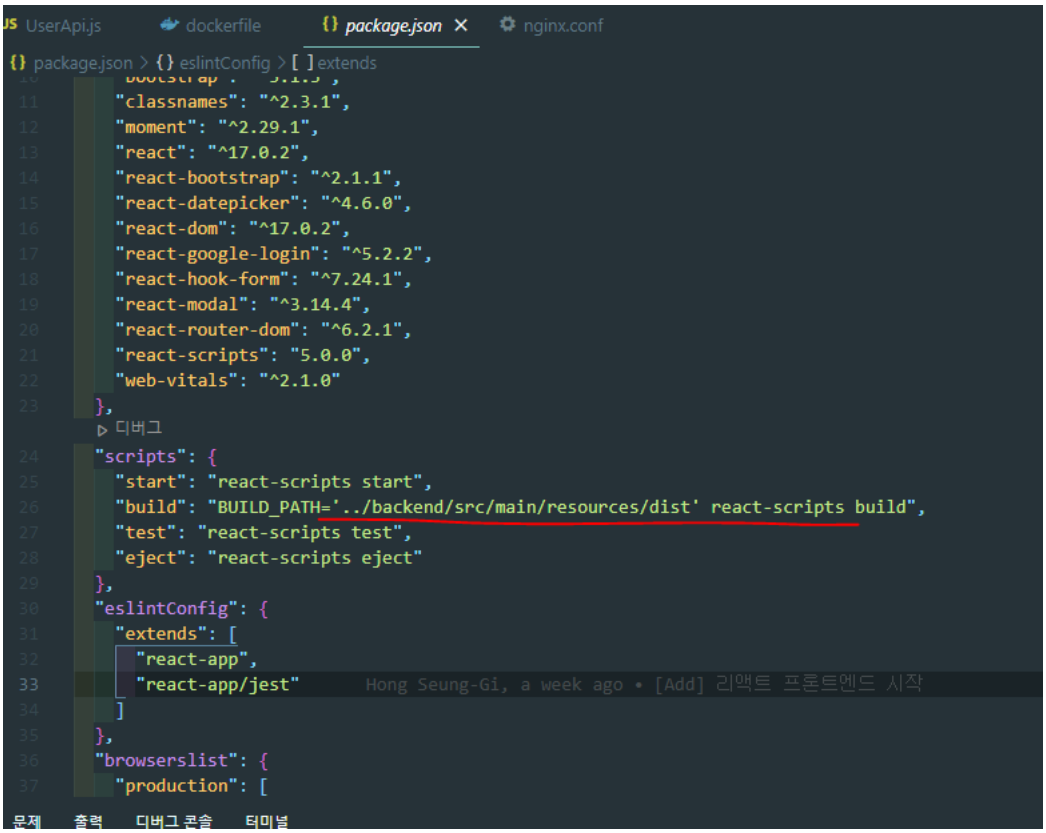
Jenkins에서 npm을 빌드시켜야 하므로 `Execute shell` 선택

알맞은 디렉토리로 이동

`npm install` 후 `npm run build` 로 build폴더를 생성시킨다.

그러면 해당 디렉토리에 `build`가 생성됨

안생겼으면 프론트 프로젝트의 `package.json`을 확인



나는 처음에 build파일 경로를 설정해주는 부분이 backend프로젝트의 classpath로 잡혀있었다.

그러므로 저 부분을 ./build 로 설정해주면 된다.

여기까지 됐으면 이 build 폴더를 nginx로 복사해주어야 한다.

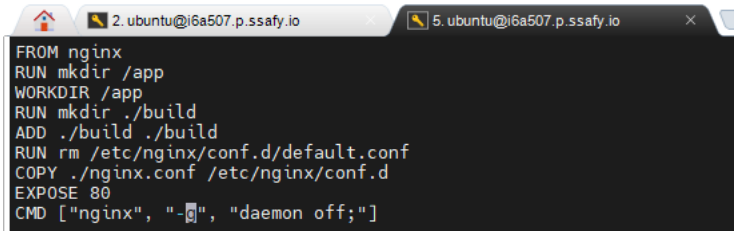
2. nginx로 build폴더 복사하기

이제부터 ssh로 작업해야 하므로 모바엑스텀으로 이동

`/home/jenkins/workspace/jjanhae-dev-frontend-build` 로 이동

이곳에 `deploy.sh` 와 `dockerfile` 을 생성해주어야 함

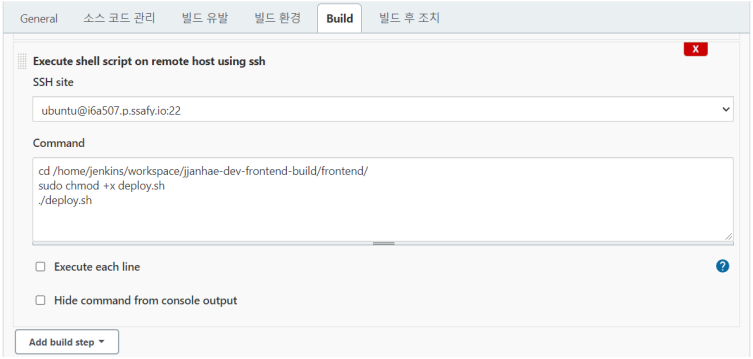
`sudo vi dockerfile`



`sudo vi deploy.sh`

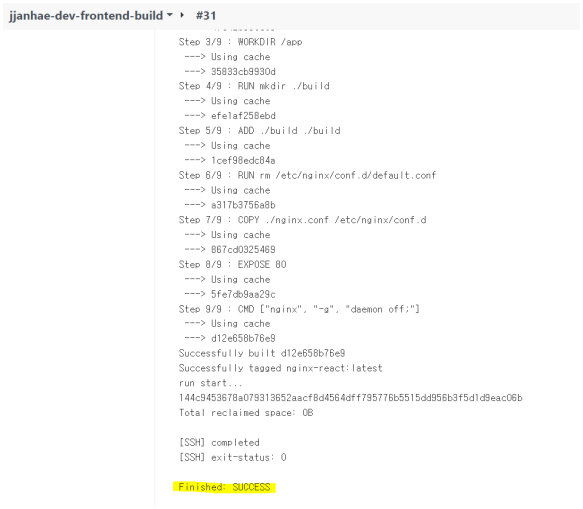
위와 같이 입력

dockerfile과 deploy.sh를 작성했으면 이제 이 배포파일을 자동 실행하도록 만들어야 한다.



다시 젠킨스로 돌아와 ssh에서 해당 파일을 실행하도록 설정한다.

저장 후 Build Now 눌러서 빌드 실행



에러없이 SUCCESS 뜨면 끝

SSL 발급

letsencrypt와 certbot을 이용해 ssl인증서를 발급한다.

reverse proxy nginx 서버를 따로 두어 react와 api를 호출하는 http 요청을 https로 리다이렉트 한다.

certbot 설치

`sudo apt-get install certbot`

reverse proxy nginx 설정

ec2에 접속

`cd /home/jjanhae/git-repository`

해당 경로 안에 git repository 복사해옴

그리고 `gateway` 폴더 생성

`gateway` 폴더 안에 다음과 같이 작성한다.

`data/nginx/conf.d/app.conf`

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

    server_name i6a507.p.ssafy.io;

    location /.well-known/acme-challenge/ {
        allow all;
        root /var/www/certbot;
    }

    location / {
```

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

`docker-compose.yml`

```
version: '3.8'
services:
  gateway:
    image: nginx
    restart: always
    ports:
      - 80:80
      - 443:443
    volumes:
      - ./data/nginx/conf.d:/etc/nginx/conf.d
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
  certbot:
    container_name: certbot
    image: certbot/certbot
    restart: always
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
  react:
    image: nginx-react
  api:
    image: jjanhae
    environment:
      - TZ=Asia/Seoul
  kurento:
    image: kurento/kurento-media-server
```

`init-letsencrypt.sh`

```
#!/bin/bash

if ! [ -x "$(command -v docker-compose)" ]; then
  echo 'Error: docker-compose is not installed.' >&2
  exit 1
fi

domains="i6a507.p.ssafy.io"
rsa_key_size=4096
data_path="./data/certbot"
email="dtd1170@naver.com" # Adding a valid address is strongly recommended
staging=0 # Set to 1 if you're testing your setup to avoid hitting request limits

if [ -d "$data_path" ]; then
  read -p "Existing data found for $domains. Continue and replace existing certificate? (y/N) " decision
  if [ "$decision" != "Y" ] && [ "$decision" != "y" ]; then
    exit
  fi
fi


if [ ! -e "$data_path/conf/options-ssl-nginx.conf" ] || [ ! -e "$data_path/conf/ssl-dhparams.pem" ]; then
  echo "### Downloading recommended TLS parameters ..."
  mkdir -p "$data_path/conf"
  curl -s https://raw.githubusercontent.com/certbot/certbot/master/certbot-nginx/certbot_nginx/_internal/tls_configs/options-ssl-ng
  curl -s https://raw.githubusercontent.com/certbot/certbot/master/certbot/certbot/ssl-dhparams.pem > "$data_path/conf/ssl-dhparams
  echo
fi

echo "### Creating dummy certificate for $domains ..."
path="/etc/letsencrypt/live/$domains"
mkdir -p "$data_path/conf/live/$domains"
docker-compose run --rm --entrypoint "\
  openssl req -x509 -nodes -newkey rsa:$rsa_key_size -days 1\
    -keyout '$path/privkey.pem' \
    -out '$path/fullchain.pem' \
```

```
    -subj '/CN=localhost'" certbot
echo


echo "### Starting nginx ..."
docker-compose up --force-recreate -d gateway
echo

echo "### Deleting dummy certificate for $domains ..."
docker-compose run --rm --entrypoint "\
  rm -Rf /etc/letsencrypt/live/$domains && \
  rm -Rf /etc/letsencrypt/archive/$domains && \
  rm -Rf /etc/letsencrypt/renewal/$domains.conf" certbot
echo


echo "### Requesting Let's Encrypt certificate for $domains ..."
#Join $domains to -d args
domain_args=""
for domain in "${domains[@]}"; do
  domain_args="$domain_args -d $domain"
done

# Select appropriate email arg
case "$email" in
  "") email_arg="--register-unsafely-without-email" ;;
  *) email_arg="--email $email" ;;
esac

# Enable staging mode if needed
if [ $staging != "0" ]; then staging_arg="--staging"; fi

docker-compose run --rm --entrypoint "\
  certbot certonly -a webroot -v --debug-challenges -w /var/www/certbot \
    $staging_arg \
    $email_arg \
    $domain_args \
    --rsa-key-size $rsa_key_size \
    --agree-tos \
    --force-renewal" certbot
echo

echo "### Reloading nginx ..."
docker-compose exec gateway nginx -s reload
```

`init-letsentcypt.sh` 실행

`./init-letsencrypt.sh`

ssl 인증서 정상발급 되면 `app.conf` 파일에 https server 설정 추가

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

    server_name i6a507.p.ssafy.io;

    location /.well-known/acme-challenge/ {
        allow all;
        root /var/www/certbot;
    }

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

server {
    listen 443 ssl;
    server_name i6a507.p.ssafy.io;
    server_tokens off;

    ssl_certificate /etc/letsencrypt/live/i6a507.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i6a507.p.ssafy.io/privkey.pem;
```

```
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://react;
        proxy_set_header    Host                $http_host;
        proxy_set_header    X-Real-IP           $remote_addr;
        proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
    }

    location /api {
        proxy_pass http://api:8081;
        proxy_set_header    Host                $http_host;
        proxy_set_header    X-Real-IP           $remote_addr;
        proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
    }
}
```

`docker-compose up --build -d nginx`

revers proxy nginx 서버 기동

오픈비두

방화벽 설정

```
sudo ufw allow ssh
sudo ufw allow 80/tcp
sudo ufw allo 443/tcp
sudo ufw allow 3478/tcp
sudo ufw allow 3478/udp
sudo ufw allow 40000:57000/tcp
sudo ufw allow 40000:57000/udp
sudo ufw allow 57001:65535/tcp
sudo ufw allow 57001:65535/udp
sudo ufw enable
#sudo ufw disable 은 비활성화
#sudo ufw status verbose 은 상태 확인
```

배포

opt, OpenVidu 둘 다 `home/ubuntu` 에 만든다!!!!!

1. 설치 권장 폴더 생성 : `mkdir opt`
2. `cd opt`
3. 설치 스크립트 다운로드 및 실행 :

   `curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | bash`
4. 적힌대로 하면 됨
   a. `cd openvidu`
   b. `nano .env`

      OPENVIDU_SECRET은 알파벳, 숫자, `-`, `_` 만 사용할 수 있다.

      ```
      DOMAIN_OR_PUBLIC_IP=i6a507.p.ssafy.io
      OPENVIDU_SECRET=jjanhae
      CERTIFICATE_TYPE=letsencrypt #letsencrypt로 했는데 selfsigned 해도 되는지 아직 차이를 잘 모르겠음 ㅠㅠ
      LETSENCRYPT_EMAIL=이메일
      ```

```
DOMAIN_OR_PUBLIC_IP=i6a507.p.ssafy.io

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

# 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=jjanhae@naver.com
```

5. `./openvidu start`
   - 처음이면 KMS(Kurento Media Server) 설치됨

```
---------------------------------------------------

   OpenVidu is ready!
   --------------------------

   * OpenVidu Server URL: https://i6a507.p.ssafy.io:5443/

   * OpenVidu Dashboard: https://i6a507.p.ssafy.io:5443/dashboard

---------------------------------------------------
```