SSH 접속

- SSH 로그인(MobaXterm 이용 이후로도 모든 과정 Mobaxterm 이용함)

```
 Session → SSH → Remote host에 도메인명 입력(k7d209.p.ssay.io) → Advanced SSH
settings에서 Use private key 선택 후 pem키 등록
```

MySQL 설치

- MySQL 설치

```
$ sudo su
$ apt-get update
$ apt-get install mysql-server
```

- MySQL 사용자 생성

```
$ sudo mysql -u root -p
mysql> cerate user 'username'@'%' identified by 'password';
```

username과 password 설정(보안 강화)

- MySQL 사용자 확인

```
mysql> show databases;
mysql> select user, host from user;
```

- DB 생성

```
mysql> create database DB;
mysql> show databases;
```

- DB 권한 부여

```
mysql> grant all privileges on DB.* to 'username'@'%';
mysql> flush privileges;
mysql> show grants for 'username'@'%';
```

%로 모든 경우 권한 부여함, 필요 시 다른 설정으로 변경

- MySQL 외부 접속 설정

```
$ cd /etc/mysql/mysql.conf.d
$ sudo vim mysqld.cnf
bind-address를 0.0.0.0으로 변경
```

- MySQL Workbench 접속

```
$ sudo service mysql restart
재시작 후 Workbench 이용해 접속
```

Docker 설치

- 사전 패키지 설치

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

- gpg키 다운로드

```
$ 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 -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

도커 설치 전 리눅스 패키지 툴이 유효한지 확인을 위해 gpg키를 통해 검증

- Docker 설치

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

Jenkins 설치(도커 컨테이너) 및 계정 생성

- Jenkins container 생성

```
$ sudo vim docker-compose.yml
```

```
# docker-compose.yml

version: '3'

services:
    jenkins:
        image: jenkins/jenkins:lts
        container_name: jenkins
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
            - /jenkins:/var/jenkins_home
        ports:
            - "9090:8080"
        privileged: true
        user: root
```

EC2 9090 포트와 container 8080포트 연결

```
$ sudo docker-compose up -d
$ sudo docker ps
$ sudo docker logs jenkins
```

container 생성 후 확인 그 이후 Jenkins 접속에 필요한 password 체크

- Jenkins 접속
    - 도메인:9090을 통해 접속
    - Create First Admin User에서 정보들을 입력 후 Save and Continue
    - Save and Finish → Start using Jenkins

- 플러그인 설치
    - Jenkins 관리 → 플러그인 관리 → 설치 가능
    - gitlab 검색
        - GitLab, Generic Webhook Trigger, Gitlba API, GitLab Authentication → Install without restart
    - docker 검색
        - Docker, Docker Commons, Docker Pipeline, Docker API → Install without restart
    - SSH 검색
        - Publish Over SSH → Install without restart

Jenkins Project WebHook 설정

- gitlab 레포지토리 생성
    - BackEnd, FrontEnd, AI

- Jenikins 페이지에서 새로운 item 클릭
    - 프로젝트 이름 작성 후 Freestyle project 클릭
    - 소스 코드 관리 탭에서 Git 클릭하고 Repository URL에 gitlab URL 입력(아직 제대로 연결이 되지 않았으므로 에러 메시지가 나타나는 것이 정상)
    - Credentials에서 +Add → Jenkins를 클릭하고 Username(gitlab 아이디), Password(gitlab 비밀번호), ID(구분 가능한 아무 텍스트)를 입력 후 Add 클릭(에러 메시지가 사라졌으면 성공)
    - 빌드 유발 탭에서 Build when a change is pushed to GitLab. 체크하고 생성되는 고급 버튼을 클릭 → 아래쪽의 Secret token에서 Generate를 클릭 후 생성되는 토큰을 저장(추후 Gitlab과 WebHook 연결할 때 사용됨)
    - Build Steps 탭 이동 → Add build step을 클릭하고, Execute Shell을 선택 → 명령어로 pwd를 입력하고 저장
    - 지금 빌드를 클릭하여 수동빌드를 진행 → 빌드 히스토리에서 Console Output에 들어가서 입력했던 명령어가 작동한 것을 확인

- Gitlab WebHook 연결
    - Gitlab Repository로 이동
    - Settings → WebHooks로 이동

- URL에 http://배포서버공인IP:9090/project/생성한jenkins프로젝트이름 입력
- Secret token에 저장해둔 값을 입력
- Tirgger로 Push events, Merge request events를 체크(상황에 따라 merge event는 안해도 됨), 대상 Branch는 master로 설정(release가 있다면 release로 설정)
- Add Webhook을 클릭하여 webhook 생성
- 생성 후 test → Push events 클릭(HTTP 200 응답이 온다면 성공)
- Jenkins에서도 응답이 잘 넘어가는지 확인

Jenkins에 Docker 설치

- Jenkins 내에 Docker 설치를 위해 jenkins bash shell에 접근

```
$ sudo docker exec -it jenkins bash
```

- 사전 패키지 설치

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

루트 계정 접속이므로 sudo 필요 없음

- gpg 키 다운로드

```
# mkdir -p /etc/apt/keyrings
# curl -fsSL https://download.docker.com/linux/debian/gpg | 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/debian \
    $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

- Jenkins os 확인

```
# cat /etc/issue
```

debian으로 나타나는 것을 확인할 수 있음

- Docker 설치

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

DockerFile 작성 및 image build

- 각각의 프로젝트 폴더에 DockerFile 작성
  - SpringBoot DockerFile

```
FROM adoptopenjdk/openjdk11
ENV JAR_FILE=build/libs/slr-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

  - React DockerFile

```
FROM node:16.15.0 as build-stage
WORKDIR /var/jenkins_home/workspace/deploy/FE/tlw/
COPY package*.json ./
RUN npm install --force
COPY . .
RUN npm run build
FROM nginx:stable-alpine as production-stage

COPY --from=build-stage /var/jenkins_home/workspace/deploy/FE/tlw/build
/usr/share/nginx/html
# COPY --from=build-stage
/var/jenkins_home/workspace/deploy/FE/tlw/deploy_conf/nginx.conf
/etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g","daemon off;"]
```

  - AI DockerFile

```
FROM python:3.7

WORKDIR /app/

COPY ./fastApi.py /app/
COPY ./ /app/

RUN pip install -r requirements.txt

CMD uvicorn --host=0.0.0.0 --port 8000 fastApi:app
```

- Jenkins에서 DockerFile로 DockerImage 생성
    - 구성 → Build 탭 → Excute shell에 작성

```
cd BE/slr
chmod +x ./gradlew
./gradlew clean build
docker build -t spring .

cd ../..
cd FE/tlw
docker build -t react .

cd ../..
cd AI
docker build -t ai .
```

    - 저장 후 빌드
    - Docker image 확인

```
$ sudo docker exec -it jenkins bash
# cd /var/jenkins_home/imgage_tar
# ls
# exit
```

Jenkins Containger 내에서 DockerImage 확인

```
$ cd /jenkins/images_tar
$ls
```

EC2의 DockerImage 확인

DockerImage로 Container 생성

- Jenkins SSH 연결 설정
    - Jenkins 관리 → 시스템 설정
    - Publish over SSH에서 SSH Servers 추가 버튼 클릭
    - Name(아무거나), Hostname(EC2 IP), Username(ubuntu) 작성 후 고급 버튼 클릭
    - Use password authentication, or use different key 체크
    - 생성된 form에 pem키 파일(VSCode 등으로 open) 내용을 넣음
    - Test Configuration 클릭 후 Success가 나오면 성공

- Jenkins 빌드 후 조치로 SSH 명령어 전송(EC2에 Docker Container 생성)
    - 구성 → Execute shell에 명령어 추가

```
cd BE/slr
chmod +x ./gradlew
./gradlew clean build
docker build -t spring .
docker ps -f name=spring -q | xargs --no-run-if-empty docker container stop
docker container ls -a -f name=spring -q | xargs -r docker container rm
docker run -d -p 8081:8081 -p 8443:8443 --name spring spring

cd ../..
cd FE/tlw
docker build -t react .
docker ps -f name=react -q | xargs --no-run-if-empty docker container stop
docker container ls -a -f name=react -q | xargs -r docker container rm
docker run -d -p 80:80 -p 443:443 --name react react

cd ../..
cd AI
docker build -t ai .
docker ps -f name=ai -q | xargs --no-run-if-empty docker container stop
docker container ls -a -f name=ai -q | xargs -r docker container rm
docker run -d -p 8000:8000 --name ai ai
```

8081과 8443번 포트(spring), 80과 443번 포트(react), 8000번 포트(AI)

Nginx 경로 설정

- 보안을 위해 SSL 설정 후 FrontEnd와 BackEnd 간의 크로스 도메인 이슈를 방지하기 위해 설정

- nginx.conf 파일 생성

```
$ cd /jenkins/workspace/deploy/FE/tlw
$ sudo mkdir deploy_conf
$ cd deploy_conf
$ sudo vim nginx.conf
```

```
upstream backend{
        ip_hash;
        server 172.26.7.81:8443;
}

server {
    listen 80;
    server_name k7d209.p.ssafy.io;
    location / {
        return 301 https://$host$request_uri;
    }

    location / {
        root /usr/share/nginx/html;
        index index.html index.html;
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_hide_header Access-Control-Allow-Origin;
        add_header 'Access-Control-Allow-Origin' '*';
        rewrite /api/(.*) /$1  break;
        proxy_pass https://backend;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $host;
    }
}
```

BackEnd는 /api로 접근 가능

https 적용(FrontEnd)

- Certbot 설치 전 사전 세팅

```
$ sudo apt-get update #apt-get
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:certbot/certbot
$ sudo apt-get update #apt-get
```

- Certbot 설치

```
$ sudo apt-get install certbot
```

- Certbot 인증

```
$ sudo certbot certonly --standalone -d [도메인명]
```

별도의 설정이 필요가 없으므로 Standalone 방식을 채택 Congratulations!가 나오면 정상 발급 완료

- 인증키 옮기기(관리를 용이하게 하기 위해서)

```
$ cd
$ sudo mkdir docker-volume
$ sudo cp /etc/letsencrypt/live/[도메인명]/fullchain.pem /home/ubuntu/docker-
volume/cert/fullchain.pem
$ sudo cp /etc/letsencrypt/live/[도메인명]/privkey.pem /home/ubuntu/docker-
volume/cert/privkey.pem
```

- Jenking 설정 변경
  - 구성 → Execute shell

```
cd BE/slr
chmod +x ./gradlew
./gradlew clean build
docker build -t spring .
docker ps -f name=spring -q | xargs --no-run-if-empty docker container stop
docker container ls -a -f name=spring -q | xargs -r docker container rm
docker run -d -p 8081:8081 -p 8443:8443 --name spring spring

cd ../..
cd FE/tlw
docker build -t react .
docker ps -f name=react -q | xargs --no-run-if-empty docker container stop
docker container ls -a -f name=react -q | xargs -r docker container rm
docker run -d -p 80:80 -p 443:443 -v /home/ubuntu/docker-
volume/cert:/usr/share/nginx/html/cert --name react react
```

```
cd ../..
cd AI
docker build -t ai .
docker ps -f name=ai -q | xargs --no-run-if-empty docker container stop
docker container ls -a -f name=ai -q | xargs -r docker container rm
docker run -d -p 8000:8000 --name ai ai
```

-v 명령어를 통해 마운팅시킴(계속 인증서를 초기화하는것은 비효율이므로)

- nginx 설정 파일 수정

```
$ cd /jenkins/workspace/deploy/FE/tlw/deploy_conf
$ sudo vi nginx.conf
```

```
upstream backend{
        ip_hash;
        server 172.26.7.81:8443;
}

server {
    listen 80;
    server_name k7d209.p.ssafy.io;
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen       443 ssl;
    listen  [::]:443 ssl;
    server_name  k7d209.p.ssafy.io;

    #access_log  /var/log/nginx/host.access.log  main;


    location / {
        root /usr/share/nginx/html;
        index index.html index.html;
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_hide_header Access-Control-Allow-Origin;
        add_header 'Access-Control-Allow-Origin' '*';
        rewrite /api/(.*) /$1  break;
        proxy_pass https://backend;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $host;
    }


    ssl_certificate /usr/share/nginx/html/cert/fullchain.pem;
    ssl_certificate_key /usr/share/nginx/html/cert/privkey.pem;
}
```

https 설정(BackEnd)

- 발급 받은 ssl 인증서를 pkc12형태로 변환(springboot는 pem형식을 지원 안함)

```
$ openssl pkcs12  -export -in fullchain.pem \
                -inkey privkey.pem \
                -out keystore.p12 -name tomcat \
                -CAfile chain.pem \
                -caname root
```

- 생성된 pkc12 파일을 local springboot project resources 폴더로 옮겨줌(권한 설정 필요할 수도 있음)
- application.yml에 설정 추가

```
server:
  # 8081은 local용
  # port: 8081
  # 8443과 ssl 설정은 서버 배포용
  port: 8443
  ssl:
    enabled: true
    key-store: classpath:keystore.p12
    key-store-type: PKCS12
    key-store-password: yourday1108
```