Docker 설치

- SSH 연결

```
$ ssh -i J7<팀코드>T.pem ubuntu@도메인
```

Pem 파일이 있는 폴더에서 다음 명령어 입력 후 yes를 입력하면(초기 접속 시) 접속 완료

- 사전 패키지 설치

  Ubuntu 환경 준비 완료

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

- GPG Key 인증

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

- docker repository 등록

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

- docker 설치

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

- docker 버전 확인

  ```
  $ docker -v
  ```

젠킨스 설치 및 접속

- docker-compose 이용 젠킨스 컨테이너 생성

  ```
  $ vim 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
```

```
$ docker-compose up -d
```

에러!

→ Command 'docker-compose' not found

```
$ sudo apt install docker-compose
$ docker-compose up -d
```

```
에러!

ERROR: Couldn't connect to Docker daemon at http+docker://localhost - is it
running?

If it's at a non-standard location, specify the URL with the DOCKER_HOST
environment variable.
```

- 도커 그룹에 사용자 추가(아직 추가되지 않은 경우), 도커 다시 시작

```
$ sudo usermod -aG docker $USER
$ sudo service docker restart
```

→ 이 방법도 아님

👉 해결(앞에 sudo를 안붙혔다.)

```
$ sudo docker-compose up
```

- 컨테이너 생성 확인

```
$ docker ps
```

- Jenkins 브라우저 접속

  `http://도메인:9090`으로 접속

  비밀번호 입력 후 `Install suggested plugins` 클릭

- Jenkins 계정 생성

  설치가 끝나면 `Create First Admin User` 폼이 뜬다.

  `Save and Finish`, `Start using Jenkins`

- 플러그인 설치
  - 설치 가능 탭에서 검색어에 `gitlab`을 입력

    `GitLab`, `Generic Webhook Trigger`, `Gitlab API`, `GitLab Authentication`을 체크하고 `Install without restart` 클릭
  - 검색어에 `docker` 입력

    같은 방식으로 `Docker`, `Docker Commons`, `Docker Pipeline`, `Docker API`
  - SSH 관련 플러그인 `Publish Over SSH`까지 설치

젠킨스 WebHook 설정

- gitlab 테스트용 레포지토리 생성

  backend, frontend 프로젝트

- 젠킨스 메인페이지에서 `새로운 item` 클릭

  프로젝트 이름을 입력하고 `Freestyle project` 선택

- `소스 코드 관리` 탭 클릭

  None에 체크되어있는 것을 Git으로 바꾸고

  `Repository URL`에 싸피깃 레포지토리 URL을 입력한다.

  (빨간 에러 메세지가 뜨는데 지금 단계에서는 정상이다.)

  - `Credentials`에서 `+ Add` 클릭

    `Add Credentials` 폼에서 Username에 싸피깃 아이디, Password에는 싸피깃 비밀번호, ID에는 Credential을 구분할 수 있는 아무 문자를 입력한다.

    입력이 끝나면 `Add`를 클릭하고 `Credentials`에서 생성한 credential을 선택

    (오류 메세지가 사라졌으면 성공)

- `빌드 유발` 탭 이동

  다음과 같이 체크하고 아래의 `고급` 버튼 클릭

  아래로 스크롤을 조금 내려 `Secret token`을 찾아 Generate 버튼을 클릭하면 입력창에 토큰이 생성된다. 이 토큰은 Gitlab과 WebHook을 연결할 때 사용되니 저장해두자.

- `Build Steps` 탭으로 이동

  `Add build step`을 클릭하고 `Execute Shell`을 선택

  명령어를 입력할 수 있는 칸이 나타나면 일단 테스트만 하는 것이기 때문에 pwd를 입력

  여기까지 완료했으면 저장

- `지금 빌드` 클릭

  일단 수동 빌드를 진행해보자.

  완료 표시가 뜨면 성공

  빌드 히스토리에서 `Console Output`에 들어가보자.

  입력했던 명령어도 잘 작동한 것을 확인할 수 있다.

- Gitlab WebHook 연결

  깃랩 레포지토리로 이동

  `Settings` → `WebHooks` 페이지로 이동

  URL에는 `http://배포서버공인IP:9090/project/생성한jenkins프로젝트이름/` 을 입력한다.

  Secret token에는 아까 젠킨스 프로젝트를 생성할 때 저장해둔 값을 입력한다.

  Trigger로 `Push events`, (`Merge request events`) 체크. 대상 Branch는 master로 설정한다.

  여기까지 완료했다면 Add Webhook 버튼을 눌러 webhook을 생성하자.

  webhook을 생성하고 나면 빌드테스트를 위해 생성된 webhook에서 test를 누르고 Push events 클릭

  응답이 잘넘어갔다면 `HTTP 200`으로 응답코드가 온다.

  젠킨스에서도 정상적으로 빌드가 수행된 것을 확인할 수 있다.

  여기까지 완료했으면 Jenkins와 Gitlab이 연결되었다. 연결된 Gitlab의 master branch에 이벤트가 발생하면, 젠킨스에서 빌드를 수행하게 된다.

젠킨스와 연결된 gitlab 프로젝트로 도커 이미지 빌드하기

젠킨스 컨테이너 안에 도커를 설치해야 한다. 도커 설치 방법은 EC2에 도커를 설치할 때와 동일하게 진행한다.

먼저 젠킨스 bash shell에 접근해보자.

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

정상적으로 접속되면 다음과 같은 화면이 나타난다. 이제 해당 환경에서 docker를 다시 설치하자.

- 사전 패키지 설치

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

root 계정으로 접속되어있기 때문에, 젠킨스 컨테이너 내부에서는 명령어에 sudo를 지워야 한다.

- GPG Key 다운로드

  젠킨스에 gpg key를 다운로드 받을 때의 변경사항입니다.

```
# 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
```

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

  젠킨스 컨테이너 내부에서 설치된 os를 체크하는 명령어를 통해 os를 확인해보면 `debian`으로 나타나는 것을 확인할 수 있습니다.

  기존 링크에서 제공한 방식은 `ubuntu os`에 대한 gpg 키를 다운로드 하는 것이기 때문에, 이를 `debian`으로 바꿔줘야 합니다. 이를 바꾸지 않으면 패키지를 찾지 못하는 에러가 발생합니다.

  기존 명령어에서 `ubuntu`로 되어있는 부분은 `debian`으로 바꿔주면 됩니다.(초반 패키지 설치 내용 참고)

- Docker 설치

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

  여기까지 진행하면 **Jenkins container에도 Docker가 설치**되었다.

- root 계정 로그아웃

```
# exit
```

SSL 인증서

- letsencrypt 설치하기

```
$ sudo apt-get update
$ sudo apt install letsencrypt
```

- 인증서 발급

```
$ sudo letsencrypt certonly --standalone -d [도메인]
```

이메일 입력 및 안내 사항에 동의 후 진행

- root 계정 로그인

```
$ sudo su
```

- 인증서 위치 폴더 이동

```
# cd /etc/letsencrypt/live/[도메인]
```

- pem을 PKCS12 형식으로 변경

```
# openssl pkcs12 -export -in fullchain.pem -inkey privkey.pem -out key.p12 -
name airpageserver -CAfile chain.pem -caname root
```

key 파일 비밀번호 입력(jenkins 계정 비밀번호와 동일)

- 인증서 복사

인증서 보관 폴더는 미리 생성하자. (예시- `/home/ubuntu/docker-volume/ssl`)

```
$ sudo cp fullchain.pem /home/ubuntu/docker-volume/ssl
$ sudo cp privkey.pem /home/ubuntu/docker-volume/ssl
$ sudo cp key.p12 /home/ubuntu/docker-volume/ssl
```

DockerFile 작성 및 이미지 생성

각 프로젝트 폴더에 DockerFile을 만들자.

- Spring boot Dockerfile

```
#backend/Dockerfile
FROM openjdk:11 as builder
COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
RUN chmod +x ./gradlew
RUN ./gradlew bootJar

VOLUME /tmp

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

EXPOSE 8080

# 배포용 properties 실행 명령어
ENTRYPOINT ["java","-jar","/app.jar","--spring.config.name=application-
prod"]
```

```
# 만약 배포용 properties를 사용하지 않는다면
# Default properties 실행 명령어
# ENTRYPOINT ["java","-jar","app.jar"]
```

- React Dockerfile

```
# frontend/Dockerfile

# React
FROM node:16-alpine as build-stage
WORKDIR /var/jenkins_home/workspace/momssok_test/front
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/momssok_test/front/build
/usr/share/nginx/html

# COPY --from=build-stage /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

이제 젠킨스에서 생성한 도커파일을 이용해서 이미지를 생성하도록 하자.

젠킨스에 들어가서 `구성` 버튼을 클릭

`Build Steps` 탭으로 이동해서 명령어 창에 아래와 같이 입력한다.

```
docker image prune -a --force
mkdir -p /var/jenkins_home/images_tar

cd /var/jenkins_home/workspace/momssok_test/backend/
docker build -t springboot .
docker save springboot > /var/jenkins_home/images_tar/springboot.tar

cd /var/jenkins_home/workspace/momssok_test/front/
docker build -t react .
docker save react > /var/jenkins_home/images_tar/react.tar

echo "----------created images!----------"
```

- 확인

  젠킨스 bash shell에 접속

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

  위와 같이 뜨는 것을 확인하고,

`exit` 명령어로 다시 ubuntu 계정으로 돌아간 뒤 아래 명령어를 입력한다.

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

(초반에 젠킨스 컨테이너를 생성할 때 `docker-compose.yml` 파일에서 공유 폴더로 aws의 `/jenkins`와 `/var/jenkins_home`을 연결했었다.)

여기까지 확인하면 **젠킨스에서 도커 이미지를 빌드하여 tar 압축파일로 생성**하는 부분까지 완료되었다.

빌드한 도커 이미지로 컨테이너 생성하기

- Jenkins SSH 연결 설정(Publish over SSH)

  Jenkins에서 AWS로 SSH 명령어를 전송하려면 AWS 인증 키(EC2 생성할 때 사용한 pem 파일)을 등록해줘야 합니다.

  젠킨스 홈페이지에서 `Jenkins 관리`를 클릭하고, 이어서 `시스템 설정`을 클릭한다.

  가장 아래까지 스크롤을 내리면 `Publish over SSH`항목이 있다. 여기서 `SSH Servers`추가 버튼을 눌러준다.

  폼이 뜨면 Name에는 아무 이름, Hostname에는 `EC2 IP`, Username(ubuntu면 ubuntu)에는 EC2 접속 계정 이름을 입력해준다. 입력이 끝나면 고급 버튼을 클릭

  다른 건 건드리지 않고 `Use password authentication, or use a different key` 에 체크한다.

  폼이 생성되면 `Key`에 값을 입력할 것이다.

  키 페어 pem 파일을 vscode로 오픈하자. 전체 내용을 복사해서 붙여넣기 한다.

  이후 `Test Configuration` 버튼을 눌렀을 때 Success가 나오면 성공이다.

- Jenkins 빌드 후 조치로 SSH 명령어 전송(EC2에 도커 컨테이너 생성)

  젠킨스 프로젝트 페이지에서 구성 버튼을 클릭한다.

  `빌드 후 조치`탭에서 `빌드 후 조치 추가`를 클릭, `Send build artifacts over SSH`를 선택한다.

  `Source files`는 컨테이너에서 aws로 파일을 전송하는 부분인데, 의미없으므로 아무거나 입력한다. `Exec command` 부분은 아래 명령어를 복사 붙여넣기 해준다.

```
sudo docker load < /jenkins/images_tar/springboot.tar
if (sudo docker ps | grep "springboot"); then sudo docker stop springboot;
fi
sudo docker run -it -d --rm -p 8443:8443 --name springboot springboot

sudo docker load < /jenkins/images_tar/react.tar
if (sudo docker ps | grep "react"); then sudo docker stop react; fi
sudo docker run -it -d --rm -p 80:80 -p 443:443 --name react react

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

저장하고 `지금 빌드`버튼을 눌러 빌드해주면 된다.

👉 위의 과정에서 에러 발생

```
ERROR: Exception when publishing, exception message [Exec exit status not
zero. Status [1]]
Build step 'Send build artifacts over SSH' changed build result to UNSTABLE
Finished: UNSTABLE
```

명령어에서 괄호를 안닫아줬던 것..

여기까지 하면 Docker와 Jenkins를 이용한 **CI/CD 자동배포 완료**

`http://서버IP:80`으로 접속하면 React를

`http://서버IP:8443`으로 접속하면 SpringBoot를 서비스하게 된다.

Nginx Proxy 설정

위의 과정까지 해도 서버는 잘 서비스 된다.

하지만 Nginx 설정을 해놓지 않으면 Https 설정을 할 때 번거로운 작업이 추가로 생길 수 있고, 만약 프론트엔드가 Https에 성공했는데 백엔드가 Https 적용에 실패하면 https → http 의 크로스 도메인 오류 때문에 백엔드 API를 불러올 수 없는 오류가 생긴다.

따라서 한 개의 포트에서 두 서비스를 구분짓는 부분이 필요하다.

Nginx설정은 기존 React와 포트가 분리되어 8443 포트를 이용해야 접속 가능한 SpringBoot를 80 포트를 통해 접속할 수 있도록 변경시켜주는 작업이다.

- nginx.conf 파일 생성

  ubuntu 계정에서 `cd /jenkins/workspace/momssok/FRONT` 명령으로 디렉토리를 이동하자. 이후 `sudo mkdir nginx` 명령어로 디렉토리를 생성하고 `cd nginx`를 이용해 이동한다. `sudo vim nginx.conf` 명령어로 `nginx.conf`파일을 생성하고 편집기로 이동한다.

```
# /jenkins/workspace/momssok/FRONT/nginx/nginx.conf

server {
    listen 80;

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

```
# /etc/nginx/sites-available/default

##
# You should look at the following URL's in order to grasp a solid
understanding
# of Nginx configuration files in order to fully unleash the power of Nginx.
# https://www.nginx.com/resources/wiki/start/
#
https://www.nginx.com/resources/wiki/start/topics/tutorials/config_pitfalls/
# https://wiki.debian.org/Nginx/DirectoryStructure
#
# In most cases, administrators will remove this file from sites-enabled/
and
# leave it as reference inside of sites-available where it will continue to
be
# updated by the nginx packaging team.
#
# This file will automatically load configuration files provided by other
# applications, such as Drupal or Wordpress. These applications will be made
# available underneath a path with that package name, such as /drupal8.
#
# Please see /usr/share/doc/nginx-doc/examples/ for more detailed examples.
##

# Default server configuration
#
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        # SSL configuration
        #
        # listen 443 ssl default_server;
        # listen [::]:443 ssl default_server;
        #
        # Note: You should disable gzip for SSL traffic.
        # See: https://bugs.debian.org/773332
        #
        # Read up on ssl_ciphers to ensure a secure configuration.
        # See: https://bugs.debian.org/765782
```

```
        #
        # Self signed certs generated by the ssl-cert package
        # Don't use them in a production server!
        #
        # include snippets/snakeoil.conf;

        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;
        server_name _;

        location / {
                proxy_pass http://localhost:3000;
        }
        location / {
                proxy_pass http://localhost:8443;
        }

        # pass PHP scripts to FastCGI server
        #
        #location ~ \.php$ {
        #       include snippets/fastcgi-php.conf;
        #
        #       # With php-fpm (or other unix sockets):
        #       fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        #       # With php-cgi (or other tcp sockets):
        #       fastcgi_pass 127.0.0.1:9000;
        #}

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        #       deny all;
        #}
}

# Virtual Host configuration for example.com
#
# You can move that to a different file under sites-available/ and symlink
that
# to sites-enabled/ to enable it.
#
#server {
#       listen 80;
#       listen [::]:80;
#
#       server_name example.com;
#
#       root /var/www/example.com;
#       index index.html;
#
#       location / {
#               try_files $uri $uri/ =404;
```

```
#       }
#}
```

```
$ sudo systemctl stop nginx
$ sudo systemctl start nginx
$ systemctl status nginx.service
```

```
# 영상 파일 크기 설정
$ client_max_body_size 5M;
```

https 적용

```
# certbot 설치
$ sudo apt-get update
$ sudo add-apt-repository ppa:certbot/certbot
$ sudo apt-get install certbot
$ sudo apt-get install python-certbot-nginx
```

👉에러

```
E: Package 'python-certbot-nginx' has no installation candidate
$ sudo apt-get install certbot python3-certbot-nginx
```

```
# certbot 실행
$ sudo certbot --nginx
$ sudo systemctl restart nginx
```

```
# /etc/nginx/sites-available/default(https 적용 후)

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

        server_name 도메인;

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

server {
    listen 443 ssl; # managed by Certbot
    listen [::]:443 ssl ipv6only=on; # managed by Certbot

    ssl_certificate /etc/letsencrypt/live/도메인/fullchain.pem; # managed by
Certbot
    ssl_certificate_key /etc/letsencrypt/live/도메인/privkey.pem; # managed
by Certbot
    # include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    # ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
```

```
    root /var/www/html;
    index index.html index.htm index.nginx-debian.html;
    server_name 도메인;

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

    location /api {
            proxy_pass http://localhost:8443;
    }

}
```