배포 매뉴얼

분류	공통
파트	BE
태그	배포
시간	
Assign	다경 송찬 찬 박

목차

▼ SSH 연결 확인

1. 다운 받은 Pem 파일이 있는 폴더로 이동하여 다음 명령어 입력

```
ssh -i J7B107T.pem ubuntu@j7b107.p.ssafy.io
```

2. Are you sure you wnat to continue connecting ? 이라는 문구가 나오면 `yes` 입력하여 접속
3. 해당 환경에서 작업 진행
   - Termius, Putty 등을 이용하여 진행 가능

▼ Docker 설치

1. 사전 패키지 설치

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

2. gpg Key 다운로드

   리눅스 패키지 툴이 프로그램 패키지가 유효한지 확인하기 위해서 설치 전에 gpg key를 통해 검증하는 과정을 거치므로 gpg key 필요

```
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
```

3. Docker 설치

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

▼ 젠킨스(컨테이너) 설치 및 계정 생성

`docker-compose.yml` 파일을 이용하여 젠킨스 컨테이너 생성

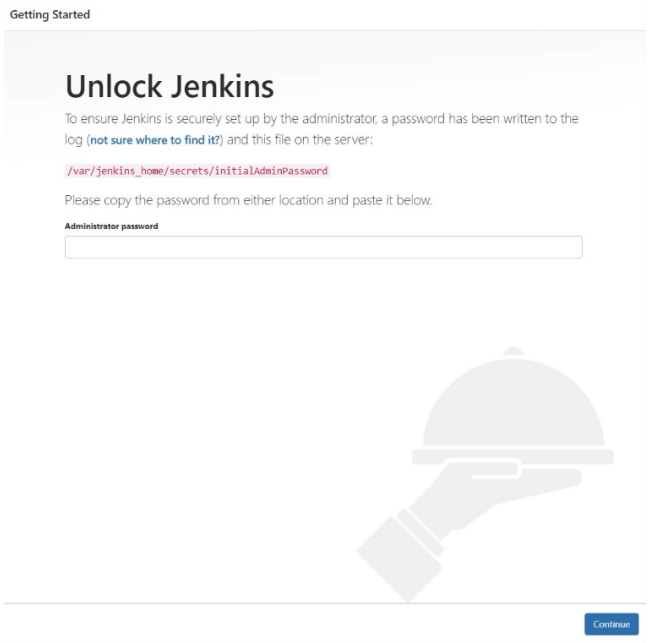
1. `docker-compose.yml` 생성

```
version: '3'

# 컨테이너 서비스
services:
    # 서비스 명(jenkins)
    jenkins:
        # 컨테이너 생성시 사용할 image
        image: jenkins/jenkins:lts
        # 생성할 컨테이너 명
        container_name: jenkins
        volumes:
            # AWS의 /var/run/docker.sock와 컨테이너 내부의 /var/run/docker.sock 연결
            - /var/run/docker.sock:/var/run/docker.sock
            # AWS의 /jenkins와 컨테이너 내부의 /var/jenkins_home 연결
            - /jenkins:/var/jenkins_home
        # 포트 매핑
        ports:
            - "9090:8080"
        # 컨테이너 시스템의 주요 자원에 연결
        privileged: true
        # 젠킨스 접속 계정 (root로 할 경우 관리자)
        user: root
```

2. `sudo docker-compose up -d` 명령어를 이용하여 컨테이너 생성
   - 컨테이너 생성 확인

     `sudo docker ps` 명령어를 통해 컨테이너 올라간 것을 확인
3. `j7b107.p.ssafy.io:9090` 으로 접속하면 다음과 같은 젠킨스 시작 화면 확인 가능



4. `sudo docker logs jenkins` 명령어를 통해 Administrator password 획득

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

91986ae95c5148a8bd98edd715f67bef

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
```

5. 획득한 password를 3번 화면 빈칸에 넣어주고 continue
6. `Install suggested plugins` 클릭하여 추천 플러그인 자동 설치
7. 젠킨스 계정 생성

▼ 젠킨스 플러그인 설치

1. `Jenkins 관리` → `플러그인 관리` → `설치가능` 으로 이동하여 아래 목록 설치

GitLab

- GitLab
- Generic Webhook Trigger
- Gitlab API
- GitLab Authentication

Docker

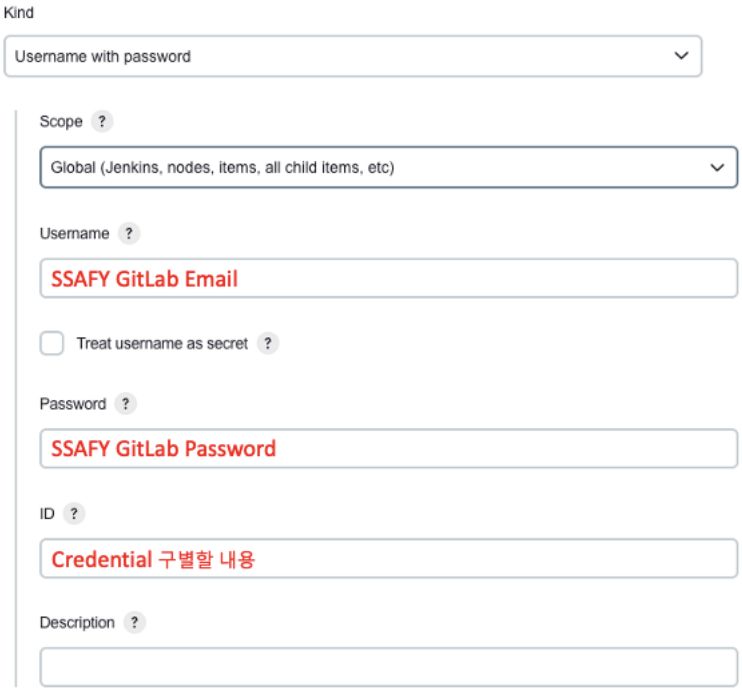
- Docker
- Docker Commons
- Docker Pipeline
- Docker API

SSH

- Publish Over SSH

▼ 젠킨스 프로젝트 생성

1. `젠킨스 메인페이지` → `새로운 Item`
   - 프로젝트 이름 입력
   - Freestyle project 선택
2. `소스코드 관리` 탭으로 이동
   - `Git 선택` → `Repository URL` 입력 (깃랩 레포 주소 입력) → `Credentials` 에서 `Add - Jenkins`



- `Credentials`에서 방금 생성한 `Credential`을 선택했을 때, 처음 나왔던 오류 메시지가 사라지면 성공

3. `빌드유발` 탭으로 이동
   - `Build when a change is pushed to GitLab. GitLab webhook URL:` `http://j7b107.p.ssafy.io:9090/project/b107` 체크
   - `고급` 버튼 클릭
   - `Secret token` Gernerate

     `Gitlab` 과 `WebHook` 연결하기 위해 `Secret token` 사용하기 때문에 따로 저장 필요
4. `Build` 탭으로 이동
   - `Add build step` → `Execute Shell` 선택
   - 연결 테스트만 진행하는 것이기 때문에 `pwd` 명령어만 입력 후 `저장`
   - 프로젝트 화면에서 `지금 빌드` 버튼을 눌러 젠킨스 수동 빌드를 진행했을 때, 왼쪽 하단에 초록색 체크가 표시되면 성공

▼ WebHook 설정, 자동 빌드 테스트

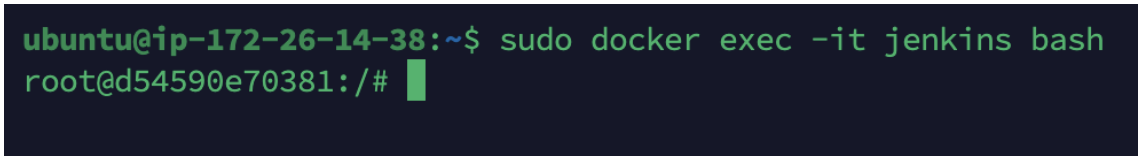
1. Gitlab Repository에서 `Settings` → `WebHooks` 로 이동
   - URL : `http://j7b107.p.ssafy.io:9090/project/생성한Jenkins프로젝트명/`
   - Secret token : `위에서 저장한 Secret toekn`
   - Trigger : `Push events(대상 Branch는 원하는 것으로 설정)` , `Merge request events`
2. `Add Webhook` 버튼을 이용하여 `webhook` 생성
3. 생성된 `webhook` 에서 `Test` → `Push events` 버튼을 이용하여 자동 빌드 테스트
4. 젠킨스 화면으로 이동하여 정상적으로 빌드가 수행되는지 확인 (초록 체크!!)

▼ GitLab 프로젝트로 Docker Image 빌드

Jenkins에서 docker build를 위해 jenkins container 내에 docker 설치 진행

1. 도커 설치를 위해 `Jenkins bash shell` 접근

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



2. `Jenkins bash shell` 에 접속한 상태에서 docker 설치

   루트 계정으로 접속되어 있기 때문에 sudo 명령어 없이 진행

   - 사전 패키지 설치

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

   - gpg Key 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/debi
         $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null
     ```

     - Jenkins container 내부 os는 `debian` 이므로 위에서 다운 받은 gpg key와 다르게 `debian` 으로 진행

   - Docker 설치

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

3. React로 만든 정적 페이지를 nginx로 돌리기 위하여 `Front` 폴더에 `nginx.conf` 파일 생성

   ```
   server {
       listen 3000;
       location / {
       root    /app/dist;
       index   index.html;
       try_files $uri $uri/ /index.html;
       }
   }
   ```

   - `nginx.conf` 파일의 위치는 `Front` 폴더 바로 아래

4. 각 프로젝트 폴더(Back, Front)에 `DockerFile` 만들기

   `모든 DockerFile`의 위치는 `Back, Front` 폴더 바로 아래

   - Back(SpringBoot)

     ```
     FROM openjdk:8-jdk-alpine AS builder

     COPY gradlew .
     COPY gradle gradle
     COPY build.gradle .
     COPY settings.gradle .
     COPY src src

     RUN chmod +x /gradlew

     RUN ./gradlew bootJar
     ```

```
FROM openjdk:8-jdk-alpine

COPY --from=builder build/libs/purgae-0.0.1-SNAPSHOT.jar app.jar

ENTRYPOINT ["java","-jar","/app.jar"]
```

- Front(React)

```
FROM node:16.15.0 as build-stage
WORKDIR /var/jenkins_home/workspace/b107/Front
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:stable-alpine as production-stage

RUN mkdir /app
WORKDIR /app
RUN mkdir ./dist

COPY --from=build-stage /var/jenkins_home/workspace/b107/Front/dist ./dist
COPY ./nginx.conf /etc/nginx/conf.d

EXPOSE 80
CMD ["nginx", "-g","daemon off;"]
```

5. Jenkins에서 DockerFile 이용하여 Docker Image 생성

- `Jenkins 프로젝트 페이지` → `구성` → `Build` → `Execute shell` 에서 아까 입력했던 `pwd` 대신 아래 내용 입력

```
docker image prune -a --force
mkdir -p /var/jenkins_home/images_tar
cd /var/jenkins_home/workspace/b107/Front/
docker build -t react .
docker save react > /var/jenkins_home/images_tar/react.tar

cd /var/jenkins_home/workspace/b107/Back/
docker build -t springboot .
docker save springboot > /var/jenkins_home/images_tar/springboot.tar

ls /var/jenkins_home/images_tar
```

  - `docker image prune -a --force` : 사용하지 않는 이미지 삭제
  - `mkdir -p /var/jenkins_home/images_tar` : Docker Image 압축파일 저장할 폴더 생성
  - `cd /var/jenkins_home/workspace/b107/Front/` : Front 폴더 경로로 이동
  - `docker build -t react .` : 도커 이미지 빌드
  - `docker save react > /var/jenkins_home/images_tar/react.tar` : 도커 이미지를 react.tar로 압축하여 폴더에 저장
  - `cd /var/jenkins_home/workspace/b107/Back/` : Back 폴더 경로로 이동
  - `docker build -t springboot .` : 도커 이미지 빌드
  - `docker save springboot > /var/jenkins_home/images_tar/springboot.tar` : 도커 이미지를 springboot.tar로 압축하여 폴더에 저장
  - `ls /var/jenkins_home/images_tar` : 해당 폴더에 있는 파일 목록 출력
- 이후 `저장` 버튼 클릭
- `Jenkins 프로젝트 메인페이지` 로 이동하여 `지금 빌드`

6. 서버에 Docker Image tar 파일들이 잘 생성되었는지 확인

- `docker-compose.yml` 파일에서 AWS의 `/jenkins` 와 컨테이너 내부의 `/var/jenkins_home` 연결하였기 때문에 각각의 폴더에 tar 파일들이 생성됨을 확인할 수 있음

```
cd images_tar
ls
```

- AWS의 `/jenkins`

```
ubuntu@ip-172-26-14-38:~$ cd /jenkins/images_tar/
ubuntu@ip-172-26-14-38:/jenkins/images_tar$ ls
env.properties  react.tar  springboot.tar  springboot_sdk.tar  springbootsdk.tar
```

- 컨테이너 내부의 `/var/jenkins_home`

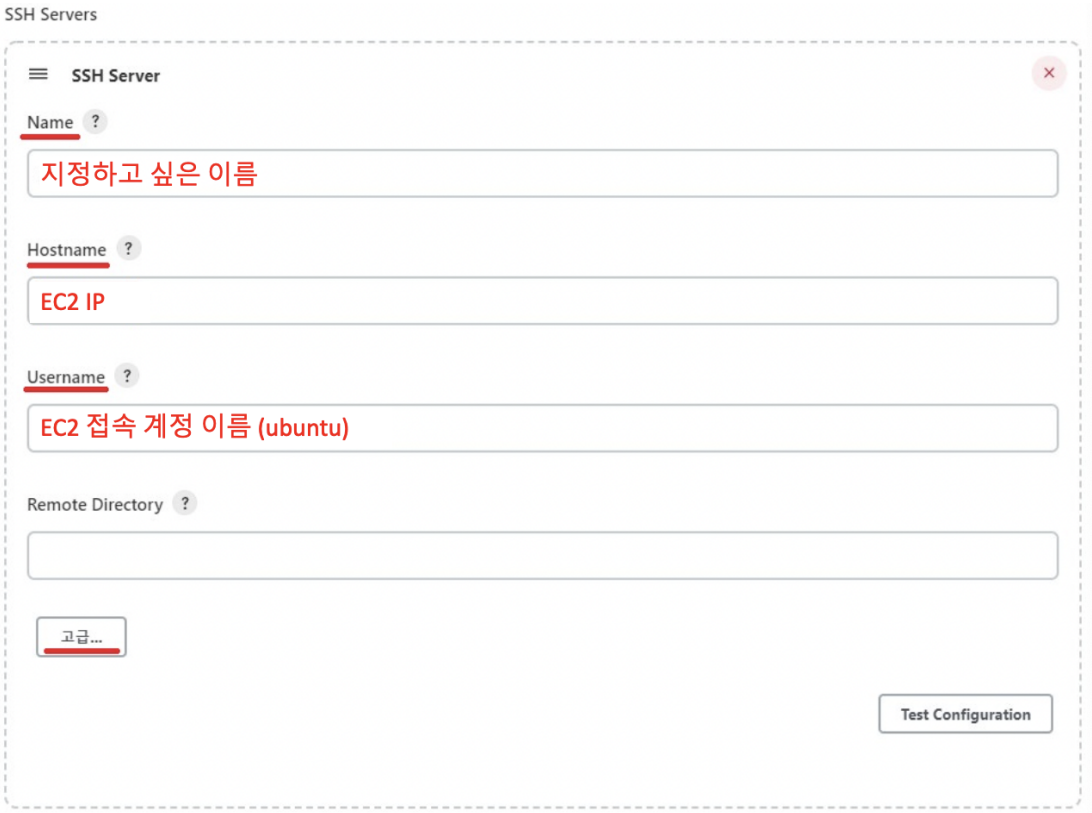
```
root@d54590e70381:/var/jenkins_home/images_tar# ls
env.properties  react.tar  springboot.tar  springboot_sdk.tar  springbootsdk.tar
```

▼ SSH 명령어 전송을 통해 빌드한 Docker Image로 Container 생성

1. Jenkins SSH 연결 설정

   젠킨스에서 AWS으로 SSH 명령어를 전송하기 위해 AWS 인증 키(EC2 생성할 때 사용한 pem 파일) 등록

- `Jenkins 관리` → `시스템 설정` → `Publish over SSH` → `SSH Servers` → `추가`



- `고급` → `Use password authentication, or use a different key` 체크 → `Key` 에 값(`EC2에서 생성했던 키 페어 pem 파일 전체`) 입력 → `Test Configuration` 클릭
  - `Success` 가 나오면 성공
  - `Auth fail` 이 나오면 실패
    - ubuntu 버전이 18.xx 버전보다 높은 경우 실패하는 경우가 있음
    - ubuntu 계정의 비밀번호를 설정하여 연결하면 됨
      - `터미널` 에서 `sudo passwd` 명령어로 root 계정 비밀번호 설정 → `successfully` 뜨면 잘 설정된 것
      - `su -` 명령어를 이용하여 root 계정으로 접속
      - 여기서 `passwd ubuntu` 명령어를 이용하여 ubuntu 계정의 패스워드 설정

- EC2에 id, pw를 이용한 로그인이 차단되어 있어 root 계정으로 접속된 상태에서 `vim /etc/ssh/sshd_config` 명령어를 통해 `sshd_config` 파일을 수정해 줌

```
# no를 yes로 바꿔주기
PasswordAuthentication no yes
```

- `service sshd reload` 명령어를 통해 sshd 재시작

  - 다시 젠킨스로 돌아와서 기존에 등록했던 `pem key` 는 다 지우기 → `Passphrase/Password` 란에 설정했던 `ubuntu 계정 패스워드` 입력 → `Test Configuration` 클릭

2. Jenkins 빌드 후 조치로 SSH 명령어 전송(EC2 Docker container 생성)

- `Jenkins 프로젝트 페이지` → `구성` → `빌드 후 조치` → `빌드 후 조치 추가` → `Send build artifacts over SSH`

`Exec command` : 주석 지우고 사용

```
# react.tar 압축해제하여 docker image로 등록
sudo docker load < /jenkins/images_tar/react.tar
# springboot.tar 압축해제하여 docker image로 등록
sudo docker load < /jenkins/images_tar/springboot.tar

# react 컨테이너가 동작중이면 stop
if (sudo docker ps | grep "react"); then sudo docker stop react; fi
# springboot 컨테이너가 동작중이면 stop
if (sudo docker ps | grep "springboot"); then sudo docker stop springboot; fi

# react라는 이름으로 컨테이너 생성 3000 포트로 연결
sudo docker run -it -d --rm -p 3000:3000 --name react react
echo "Run Front"
# springboot라는 이름으로 컨테이너 생성 8081 포트로 연결
sudo docker run -it -d --rm -p 8081:8081 --network spring  --name springboot springboot
echo "Run Back"
```

3. 지금 빌드 눌러서 빌드하기

▼ Nginx 설정과 ssl 인증서 발급 및 적용

1. `nginx` 다운로드

```
# nginx 다운
sudo apt-get install nginx

# 설치 확인 및 버전 확인
nginx -v
```

2. `letsencrypt` 다운로드

```
sudo apt-get install letsencrypt
sudo systemctl stop nginx
sudo letsencrypt certonly --standalone -d 도메인
```

3. `Congratulations!` 로 시작하는 문구가 보이면 인증서 발급 완료
4. `/etc/nginx/sites-available` 로 이동 → 적절한 이름(purgae.conf)의 파일 생성
`purgae.conf`

```
server {

        server_name purgae.net;

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

        }

        location /api {
                proxy_pass http://localhost:8081/api;
        }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/purgae.net/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/purgae.net/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
}

server {
    if ($host = purgae.net) {
        return 301 https://purgae.net$request_uri;
    } # managed by Certbot

        listen 80;
        server_name purgae.net;
    return 404; # managed by Certbot
}
```

5. 다음 명령어 차례로 실행

```
sudo ln -s /etc/nginx/sites-available/[파일명] /etc/nginx/sites-enabled/[파일명]

# 다음 명령어에서 successful이 뜨면 nginx를 실행할 수 있다.
sudo nginx -t

sudo systemctl restart nginx
```

6. 이렇게 실행하면 http로 80포트 접근시, 443 포트(https)로 리다이렉트