멍! 더 랜 드

PORTING MANUAL

목 차

I. 빌드 및 배포

1. 개발 환경

Server : AWS EC2 Ubuntu 20.04 LTS
Visual Studio Code : 1.70.0
IntelliJIDEA : 2022.1.3(Ultimate Edition) 11.0.15 + 10-b2043.56 amd64
JVM: 11.0.16+8-post-Ubuntu-0ubuntu120.04
Docker: 20.10.18
Node.js: 18.7.0
MySQL:8.0.30-1.el8
Redis: 7.0.5
Nginx: 1.18.0
Jenkins: 2.361.2

2. 설정 파일 목록

React
- .env: /jenkins/workspace/frontend/frontend/mtld
- Dockerfile: /jenkins/workspace/frontend/frontend/mtld

Spring
- application.yml :
  /jenkins/workspace/backend/backend/src/main/resources
- Dockerfile: /jenkins/workspace/backend/backend
- deploy.sh: /jenkins/workspace/backend/backend

Django
- requirements.txt : /jenkins/workspace/bigdata/bigdata/
- Dockerfile : /jenkins/workspace/bigdata/bigdata/

Docker
- docker-compose.yml: /home/ubuntu

Nginx
- test.conf: /etc/nginx/sites-enabled

3. 설정 파일 및 환경 정보

React

- .env:

```
WDS_SOCKET_PORT=0

REACT_APP_BASE_URL= REST API 요청 URL

REACT_APP_KAKAO_CLIENT_ID= 카카오 클라이언트 아이디
REACT_APP_KAKAO_CLIENT_SECRET= 카카오 클라이언트 SECRET
REACT_APP_KAKAO_REDIRECT_URI= 카카오 리다이렉트 URI
```

- Dockerfile:

```
FROM node:alpine
WORKDIR /usr/src/app
COPY ./package* /usr/src/app/
RUN npm install --legacy-peer-deps
COPY ./ /usr/src/app/
CMD ["npm","run","start"]
```

Spring

- application.yml :

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://j{DB주소}/{스키마}?serverTimezone=UTC&characterEncoding=UTF-8
    username: {USER}
    password: {USER PASSWORD}
  mvc:
    path match:
      matching-strategy: ant_path_matcher
  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true;
        use_sql_comments: true
```

```
servlet:
  multipart:
    max-file-size: 20MB
    max-request-size: 500MB
security:
  oauth2:
    client:
      provider:
        kakao:
          authorization-uri: {KAKAO AUTHORIAZTION URI}
          token-uri: {KAKAO TOKEN URI}
          userInfo-uri: {KAKAO USERINFO URI}
          user-name-attribute: id
      registration:
        kakao:
          client-id: { KAKAKO CIENT ID }
          client-secret: { KAKAKO SECRET }
          client-authentication-method: POST
          authorization-grant-type: authorization_code
          redirect-uri: { KAKAKO URI ID }
          scope:
            - profile_nickname
            - account_email
            - gender
            - age_range
          client-name: Kakao
redis:
  host: {SERVER ADDERESS _ AWS EC2}
  port: 6379

batch:
  initialize-schema: always
  job.enabled: false


logging.level:
 org.hibernate.SQL: debug
```

```
server.error.include-message: always
jwt:
 secret: {jwt SECRET}

cloud:
 aws:
   credentials:
     accessKey: {AWS ACCESSKEY}
     secretKey: {AWS SECRETKEY}
   stack:
     auto: false
   s3.bucket: {BUCKET NAME}
   region:
     static: ap-northeast-2
```

- DockerFile

```
FROM openjdk:11-jdk
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "-Duser.timezone=Asia/Seoul", "/app.jar"]
```

- deploy.sh

```
echo '실행 시작'
 echo 'git pull'
echo 'jar 파일 삭제'
 rm build/libs/*.jar
echo '빌드 시작'
 ./gradlew build -x test
echo '컨테이너 중지'
 docker stop springbootapp
echo '컨테이너 삭제'
 docker rm springbootapp
echo '도커파일 이미지 빌드'
 docker build -t springbootapp .
echo '컨테이너 실행'
 docker run -p 8080:8080 --name springbootapp —network ubuntu_default
-d springbootapp
```

Django

- DockerFile

```
FROM python:3.10.6
RUN apt-get update \
    && rm -rf /var/lib/apt/lists/*
WORKDIR /usr/src/app
COPY requirements.txt ./

RUN pip install --upgrade pip
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "manage.py", "runserver", "0.0.0.0:7777"]
```

- requirements.txt

pip freeze > requirements.txt 현재 장고에 설치된 라이브러리 추출 내용

Docker

- Docker-compose.yml

```
version: "3"
services:
    mysql:
        image: mysql    # mysql 이미지
        container_name: mysql     # 컨테이너 이름 설정
        environment:
            MYSQL_DATABASE: mtld    # 기본 데이터베이스 설정
            MYSQL_ROOT_PASSWORD: mtld106@#     # 루트 패스워드 설정
        volumes:
            - /mydb:/var/lib/mysql    # mysql 들어갈 볼륨(논리적 분리)
        ports:
            - 3306:3306    # 포트번호
    jenkins:
        image: jenkins/jenkins:lts    # 젠킨스 이미지 (lts 가장 최신 안정버전)
        container_name: jenkins    # 컨테이너 이름 설정
        volumes:     # 볼륨 설정
            - /var/run/docker.sock:/var/run/docker.sock
            - /jenkins:/var/jenkins_home
```

```
    ports:
      - "9090:8080"      # 포트 설정 (9090 부분을 자기 하고 싶은 포트번호로)
    privileged: true
    user: root
```

NGINX

- test.conf

```
server {
  listen 80;
  server_name 도메인 www.도메인;

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

server {
  listen 443 ssl;
  server_name 도메인;
  access_log off;
  # SSL 인증서 적용
  ssl_certificate
  /etc/letsencrypt/live/도메인/fullchain.pem;
  ssl_certificate_key
  /etc/letsencrypt/live/도메인/privkey.pem;

  # 프론트엔드
  location / {
  proxy_pass http://도메인:3000;
  proxy_set_header Host $host:$server_port;
  proxy_set_header X-Forwarded-Host $server_name;
  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 Upgrade $http_upgrade;
  proxy_set_header Connection "upgrade";
  proxy_redirect off;
  }
```

```
    # 벡엔드 (REST API)
    location /api/ {
    proxy_pass http://도메인:8080/;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-Host $server_name;
    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 Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_redirect off;
    # add_header 'Access-Control-Allow-Origin' '*' always;
   # add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, OPTIONS' always;
   # add_header 'Access-Control-Allow-Headers' 'content-type, authorization, x-requested-with' always;
    }
}
```

4. Docker 설치

 # docker 설치하기 위한 패키지 미리 설치

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

 # gpg 키 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 \ (엔터)
$
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/doc
ker.gpg] https://download.docker.com/linux/ubuntu  \ $(lsb_release -cs) stable" |
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- 리눅스 패키지들이 프로그램 패키지가 유효한지 확인하기 위해
  설치 전 gpg 키로 검증하기 때문에 다운로드

 # Redis 설치

$ sudo docker pull redis:latest

- redis 최신 버전으로 docker 이미지 받아오기

$ sudo docker network create redis-net

- 도커에 네트워크 생성

$ sudo docker network ls

- 네트워크 리스트 확인

$ sudo docker run --name redis -p 6379:6379 —network redis-net
-v /home/ubuntu/redisvolume -d redis:latest
redis-server --appendonly yes

- 볼륨 잡아서 컨테이너 실행

$ docker run -it --network redis-net --rm redis redis-cli -h redis

- redis cli bash 진입(접속)

 # Redis ReadOnly 에러 방지

$ sudo docker exec -it redis /bin/bash

- redis bash 진입 (컨테이너 접속)

$ redis-cli -h 127.0.0.1 -p 6379 info

- redis에 설정되어 있는 정보들 확인

$ redis-cli -h 127.0.0.1 -p 6379 config set slave-read-only no

- readonly 설정 yes -〉 no

 # 실행되고 있는 컨테이너 확인

$ sudo docker ps

 # 실행되지 않는 컨테이너 포함 모든 컨테이너 확인

$ sudo docker ps -a

 # 컨테이너 실행

$ sudo docker start [컨테이너명 또는 컨테이너ID]

 # 컨테이너 중지

$ sudo docker stop [컨테이너명 또는 컨테이너ID]

 # 컨테이너 삭제 (컨테이너가 중지되어 있을 때 가능)

$ sudo docker rm [컨테이너명 또는 컨테이너ID]

 # 이미지 확인

$ sudo docker images

 # 이미지 삭제

$ sudo docker rmi [이미지명 또는 이미지ID]

 # 컨테이너에 접속 (해당 컨테이너로 들어가서 명령어 입력 가능 환경)

$ sudo docker exec -it [컨테이너명 또는 컨테이너ID] /bin/bash

 # 컨테이너 로그 확인

$ sudo docker logs [컨테이너명]

5. SSL 인증서 설치 및 NGINX https 설정

 # nginx 실행

$ sudo service nginx start

 # nginx 종료 후 재실행

$ sudo service nginx restart

 # 수정된 설정 파일 적용 후 nginx 실행

$ sudo servcie nginx reload

 # nginx 중지

$ sudo service nginx stop

 # nginx 설정파일의 문법이 올바른지 확인

$ nginx -t

 # Nginx 설치

$ sudo apt-get update

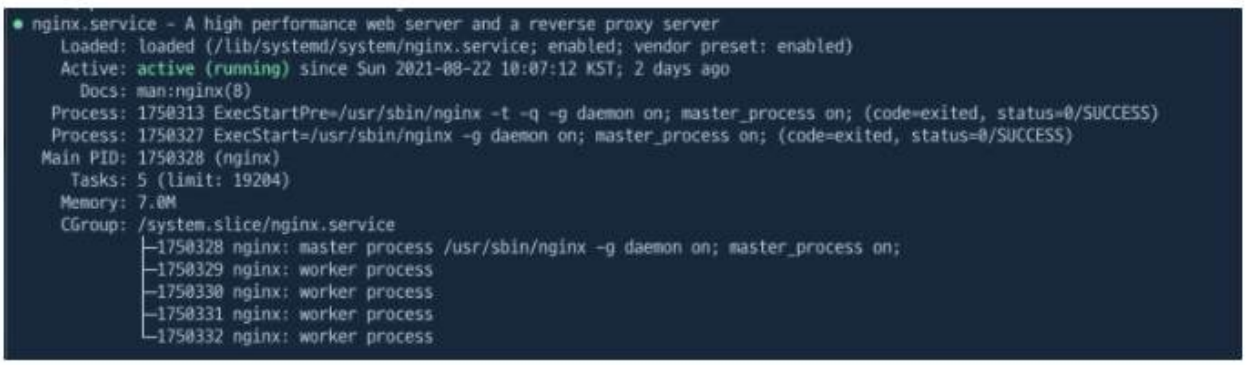
$ sudo apt install nginx -y

$ nginx -v

- nginx 버전 확인

$ sudo service nginx status

- nginx 상태 확인 (inactive 라면 실행 중인 상태)
- 꺼져있다면 sudo service nginx start로 실행



 # Certbot

- 과거에는 SSL 유료
  하지만 최근 들어, https 확산을 위해 letsencrypt 형태로 SSL/TLS 인증서를 무료로 제공
  이를 제공하는 라이브러리가 certbot
  .conf 파일을 작성 후 certbot 적용시 추가적인 인증서 설정을 해준다.
- SSL / TLS
  응용계층과 전송계층 사이에서 보안 채널을 형성해주는 역할을 수행 보안용 프로토콜
  회원가입등의 민감 정보들이 오고가기 때문에 이에 대한 암호화작업을 해야 함
- 통신이 이루어질 때마다 민감 정보 데이터를 암호화하는 방법도 있겠지만 번거롭기에
  주로 SSL/TLS를 사용, SSL은 TLS의 이전의 프로토콜

 # certbot 설치

```
$ sudo add-apt-repository ppa:certbot/certbot
$ sudo apt-get update
$ sudo apt-get install python3-certbot-nginx
$ certbot certonly --nginx -d [도메인명 ex)j7a106.p.ssafy.io]
```

- 관리자 이메일 주소 입력 (형식상 - 사실 아무거나 상관 X)
- 약관 동의 Agree
- 재단 이메일에 소식 공유 동의 No

```
$ ls -al /etc/letsencrypt/live/[도메인명]
total 12
drwxr-xr-x 2 root root 4096 Aug 21 23:03 .
drwx------ 3 root root 4096 Aug 21 23:03 ..
-rw-r--r-- 1 root root  692 Aug 21 23:03 README
lrwxrwxrwx 1 root root   41 Aug 21 23:03 cert.pem -> ../../archive/example.com/cert1.pem
lrwxrwxrwx 1 root root   42 Aug 21 23:03 chain.pem -> ../../archive/example.com/chain1.pem
lrwxrwxrwx 1 root root   46 Aug 21 23:03 fullchain.pem -> ../../archive/example.com/fullchain1.pem
lrwxrwxrwx 1 root root   44 Aug 21 23:03 privkey.pem -> ../../archive/example.com/privkey1.pem
$ sudo certbot renew —dry-run
```

- niginx 기본 파일

/etc/nginx/nginx.conf

Nginx에 관련한 설정파일로 Nginx 설정에 관한 블록들이 작성되어 있다.
이 파일에서 sites-enabled 폴더에 있는 파일들을 include하여 파일들을 가져온다.

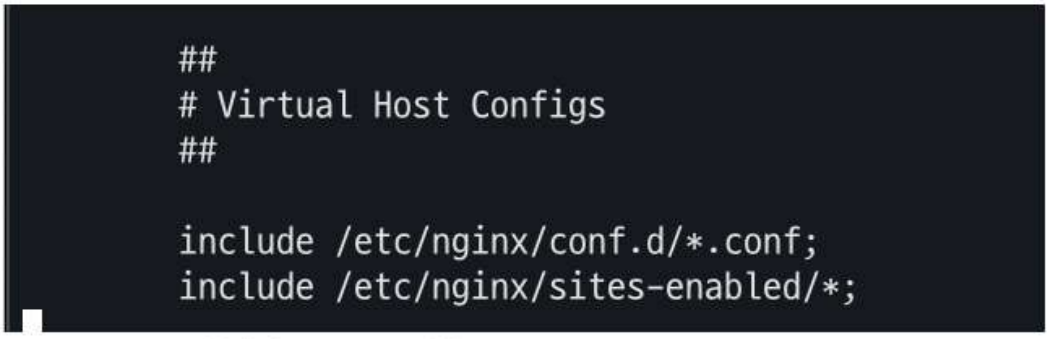
/etc/nginx/sites-available

가상 서버 환경들에 대한 설정 파일들이 위치하는 디렉토리이다.
실제 가상 서버가 돌던 안 돌던 가상 서버와 관련된 설정 파일들은 여기에 놓도록 한다.

- /ets/nginx/sites-enabled

sites-available에 있는 가상 서버 파일들중에서 실행시키고 싶은 파일을
symbolic link로 연결한 폴더이다.
이 폴더에 위치한 가상서버 환경 파일들을 읽어서 실제 서버를 세팅한다.

```
$ cd /etc/nginx
$ sudo nano nginx.conf
```



/etc/nginx/conf.d 아래에 있는 모든 .conf 파일 include
/etc/nginx/siteds-enabled 아래에 있는 모든 파일 include

- 직접 nginx.conf에 설정정보 수정/추가 가능
  - 1번은 기본설정을 건드리지 않음
  - 2번은 가상 서버 환경을 만들어 심볼릭 링크로 연결 → 2번으로 진행
    - 심볼릭 링크
    - 링크를 연결해 원본 파일을 직접 사용하는 것과 같은 효과를 내는 링크이다.
      윈도우의 바로가기와 비슷한 개념이라고 생각하면 된다.
      특정 폴더에 링크를 걸어 원본파일을 사용하기 위해 사용한다.
- cd /etc/nginx/sites-available 로 이동하여 **test.conf** 파일 작성 (이름은 상관 X)

```
server {
        listen 80;
        server_name 도메인 www.도메인;

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

    server {
        listen 443 ssl;
        server_name 도메인;
        access_log off;

        # SSL 인증서 적용
        ssl_certificate
        /etc/letsencrypt/live/도메인/fullchain.pem;
        ssl_certificate_key
        /etc/letsencrypt/live/도메인/privkey.pem;

        # 프론트엔드
        location / {
        proxy_pass http://도메인:3000;
        proxy_set_header Host $host:$server_port;
        proxy_set_header X-Forwarded-Host $server_name;
        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 Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_redirect off;
        }
```

```
    # 벡엔드 (REST API)
    location /api/ {
    proxy_pass http://도메인:8080/;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-Host $server_name;
    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 Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_redirect off;
    # add_header 'Access-Control-Allow-Origin' '*' always;
   # add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, OPTIONS' always;
   # add_header 'Access-Control-Allow-Headers' 'content-type, authorization, x-requested-with' always;
    }
}
```

 # test.conf nginx 문법 확인

```
$ sudo nginx -t
        - 만약 문법이 틀릴 시 들여쓰기나 오타 확인 : 몇번째 줄에서 오류가 있는지 알려줌
```

 # test.conf 심볼릭 링크 sites-enabled에 심볼릭 링크 작성

```
$ sudo ln -s /etc/nginx/sites-available/test.conf /etc/nginx/sites-enabled
$ cd /etc/nginx/sites-enabled
        - 경로 이동
$ ls -l
        - 심볼릭 링크 확인
        total 0
        lrwxrwxrwx 1 root root 34 Aug 21 22:18 default -> /etc/nginx/sites-available/default
        lrwxrwxrwx 1 root root 36 Aug 21 23:58 test.conf -> /etc/nginx/sites-available/test.conf
```

 # nginx 재구동/재시작

```
$ sudo service nginx reload
$ sudo service nginx restart
```

6. DB 및 Infra배포

젠킨스 실행

```
$ sudo docker run -d -p 9090:8080 -v /jenkins:/var/jenkins_home
  --name jenkins -u root jenkins/jenkins:lts
```

최초 실행후 admin 시크릿 토큰(admin secret Token) 확인

```
$ docker logs jenkins
```

docker logs jenkins 로 확인한 시크릿 토큰

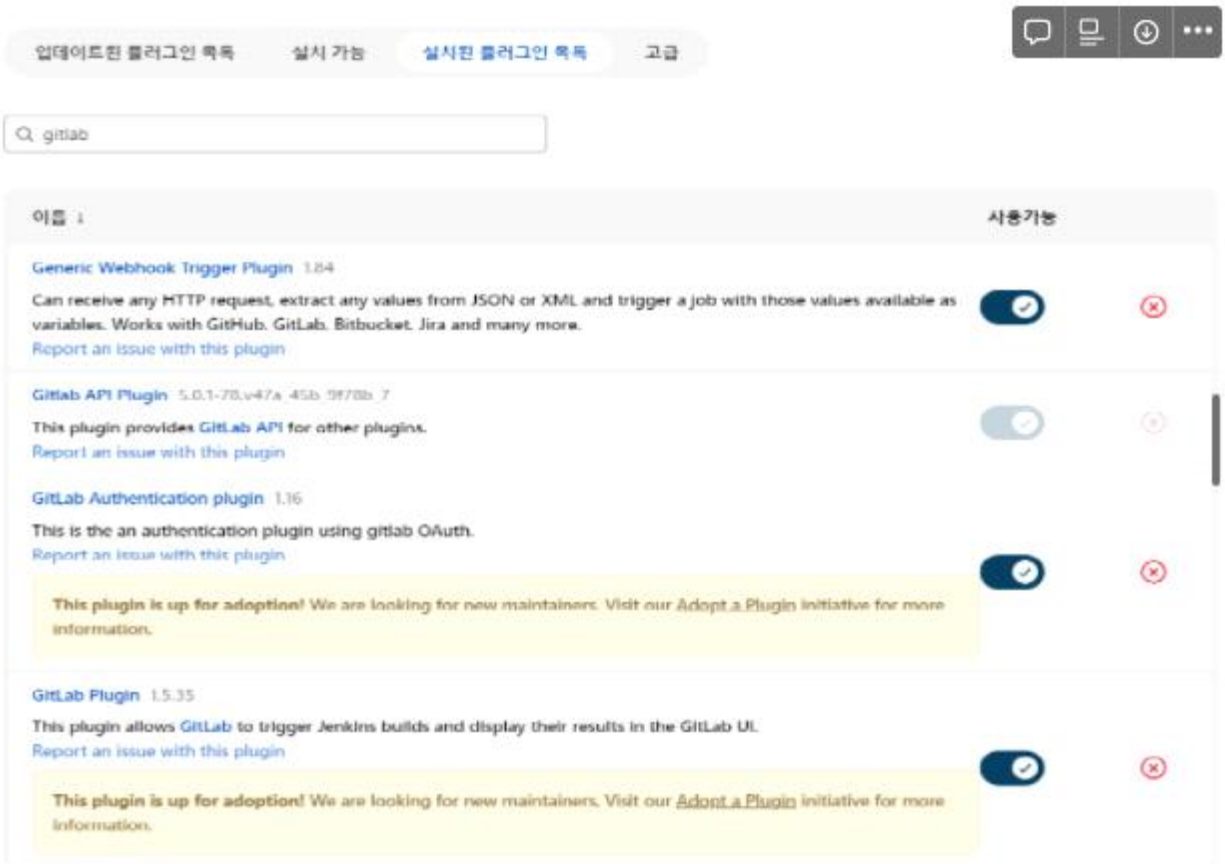
젠킨스 접속 – 탄력적IP:9090 의 첫화면에 시크릿 토큰 복사하여 붙여넣기

생성할 관리자 계정 정보 입력 〈save and continue〉

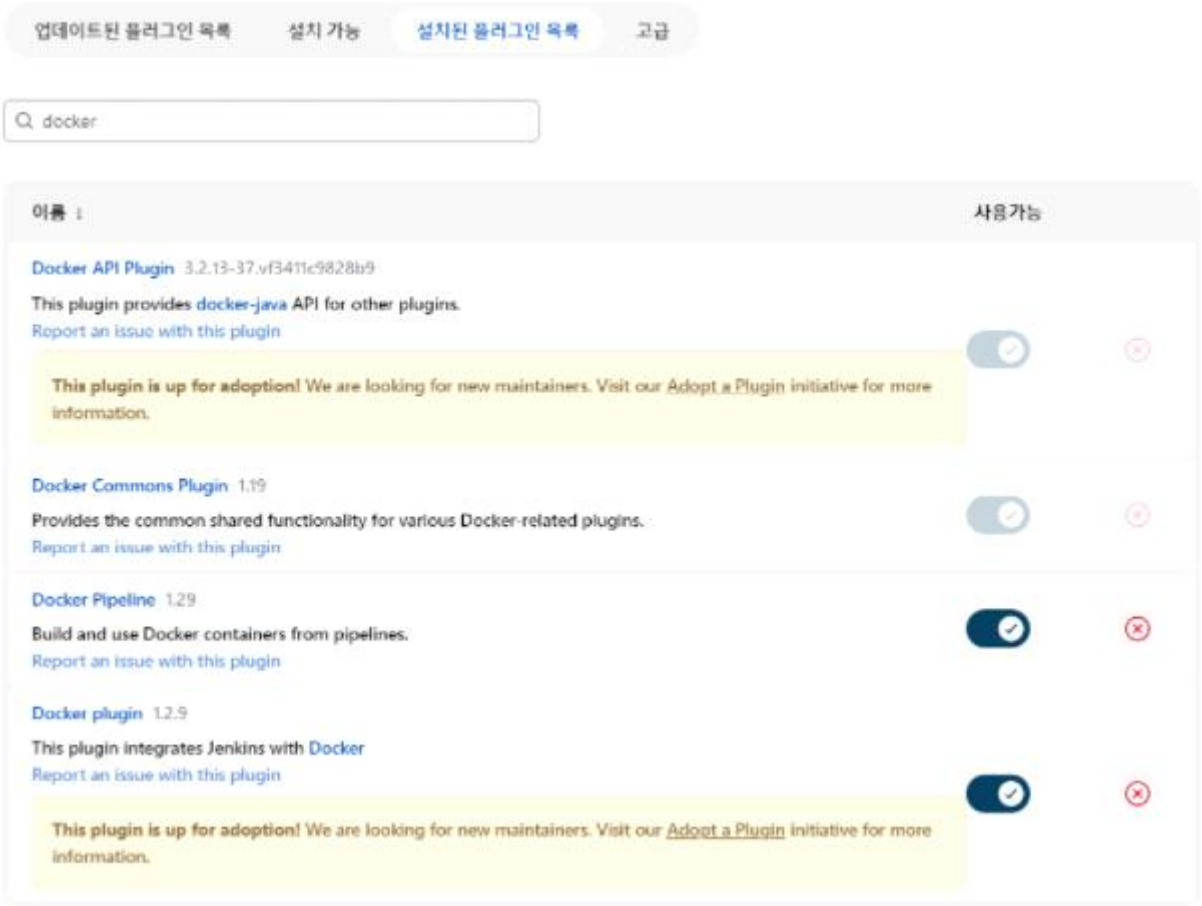
– 추가 플러그인 설치

왼쪽 DashBoard–〉 Manager JenKins ––〉 Plugin Manager → 설치가능

gitlab 검색 및 해당 플러그인 설치



docker 검색 및 해당 플러그인 설치



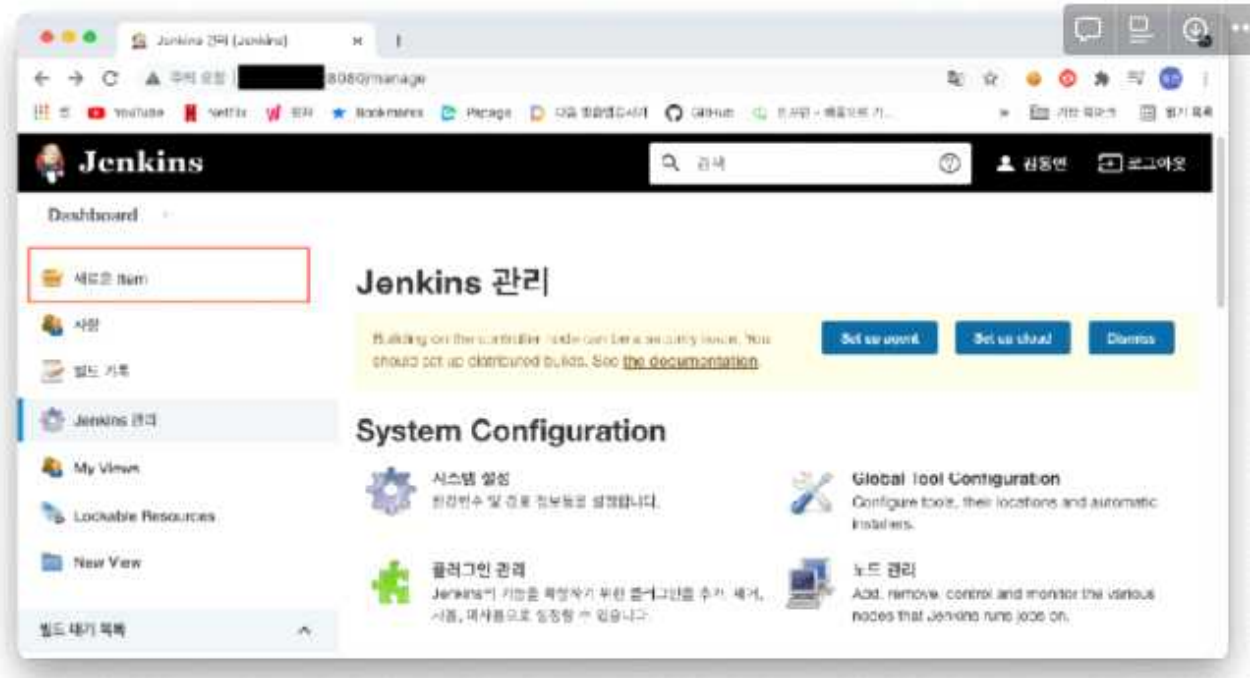
 # 젠킨스 접속해서 docker 설치

```
$ sudo docker exec -it jenkins /bin/bash
$ apt-get update -y
$ apt-get install -y
$ apt-get install docker.io -y
        - docker 설치
$ docker -v
        - docker 버전 확인
```

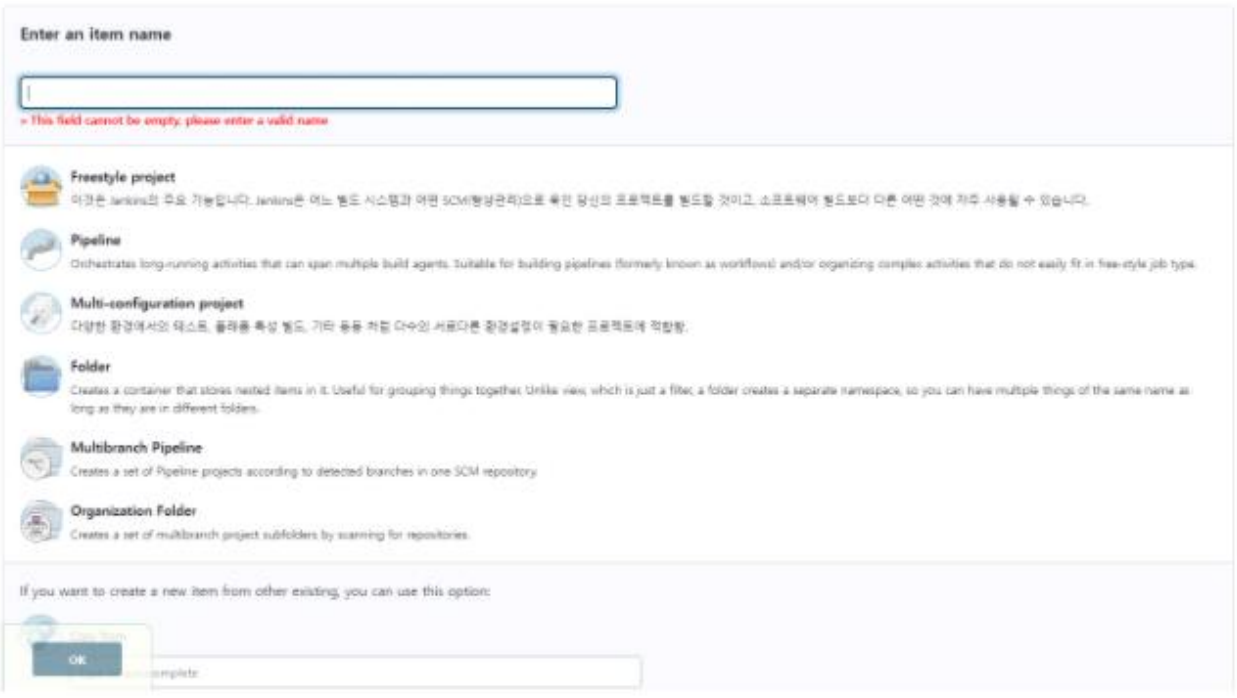
7. Backend CI/CD

젠킨스 접속

왼쪽 -〉 Dashboard --〉 새로운 Item

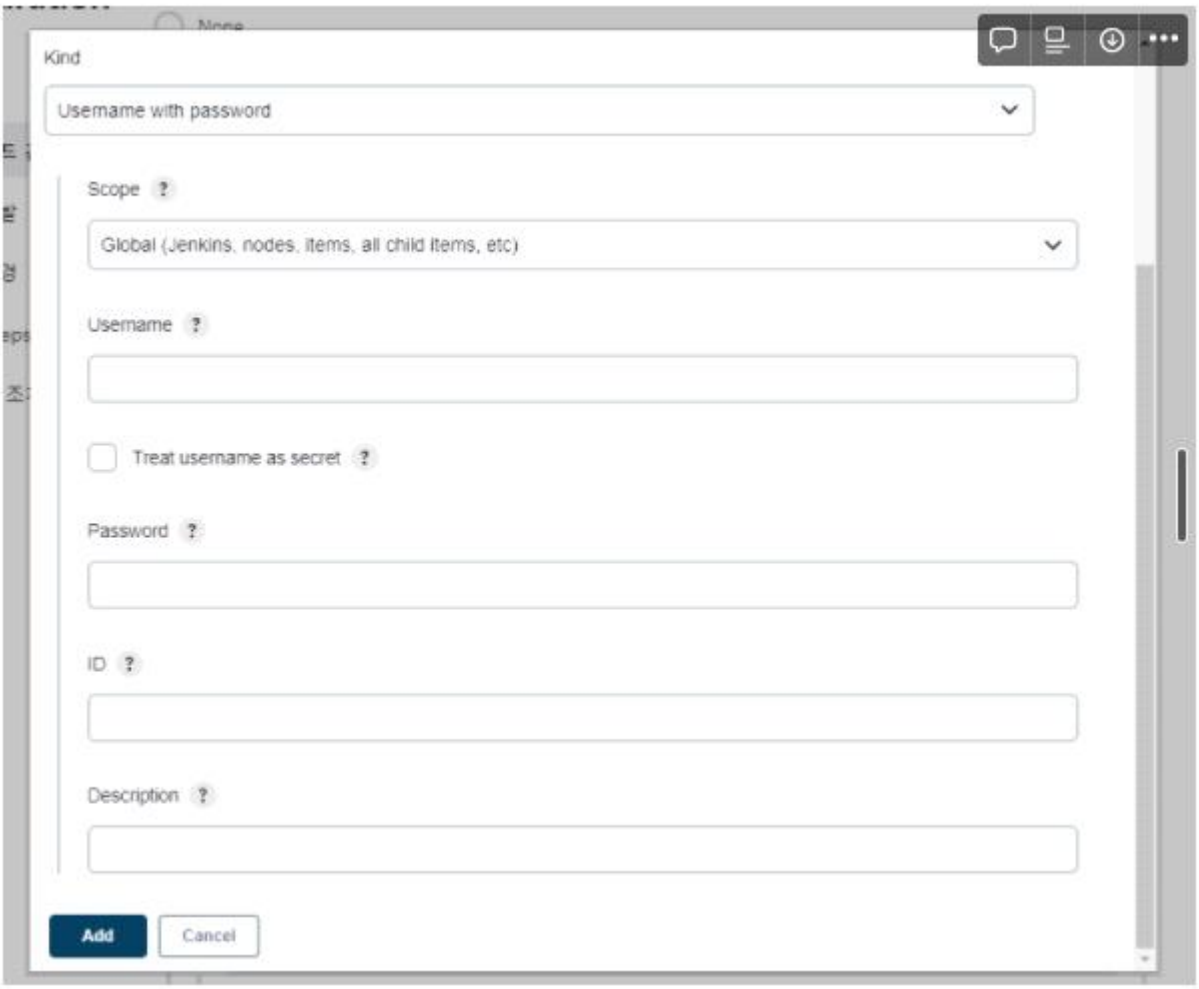


Free Style project / name : backend로 설정 → OK



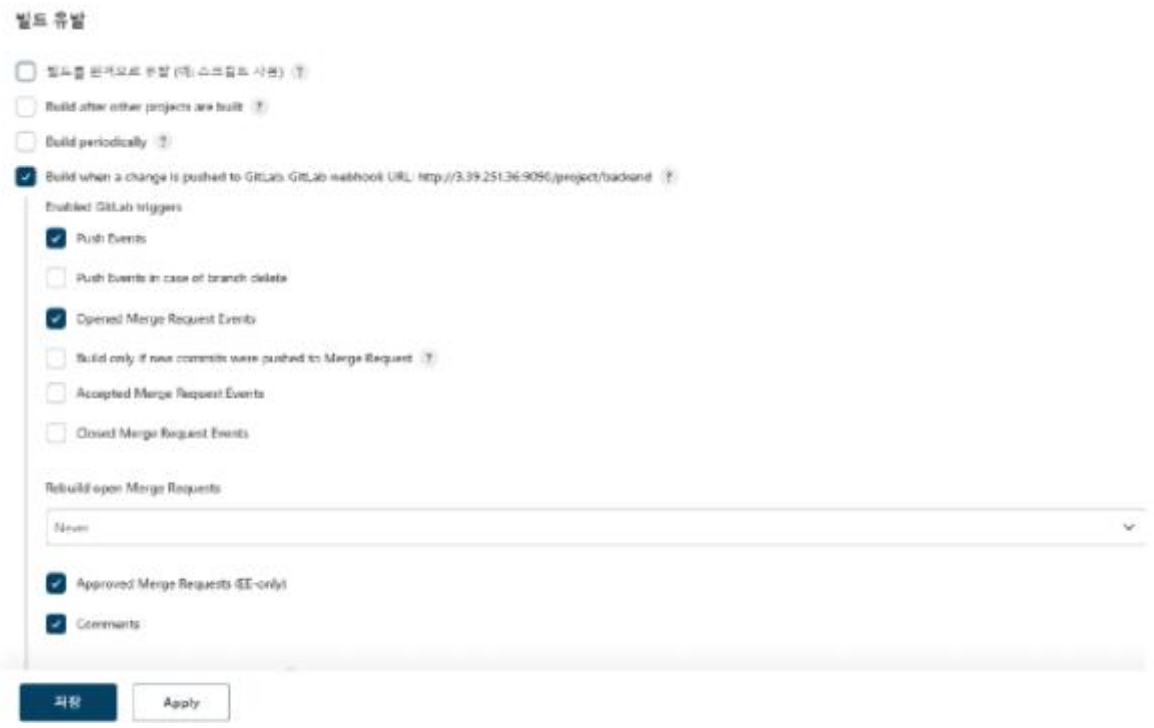
- 소스코드 관리 → git 선택
  - Repositories URL : gitlab repository HTTPS Clone
  - Credential → Add → Jenkins

  - domain : Global credentials
  - kind : Username with password
  - username : gitlab ID (리포지터리 접근 권한 있어야함)
  - password : gitlab password



- Branches to build
  - backend 브랜치
  (backend의 commit 또는 Merge를 했을 때 영향받는 브랜치)
  (ex. dev로 했을 시, BE 변경사항이 없다 하더라도 그 외 변경사항이 생기면 빌드함)

- 빌드 유발
  - 4번째 선택 _ build when a change~~
  - push, optional merge request, approved Merge Request, commets 선택



고급 선택 – secter token generate

– Build – Execute Shell

```
echo "======jenkins build 백엔드 started...========"
pwd
cd backend
chmod +x gradlew
chmod +x ./deploy.sh
./deploy.sh
```

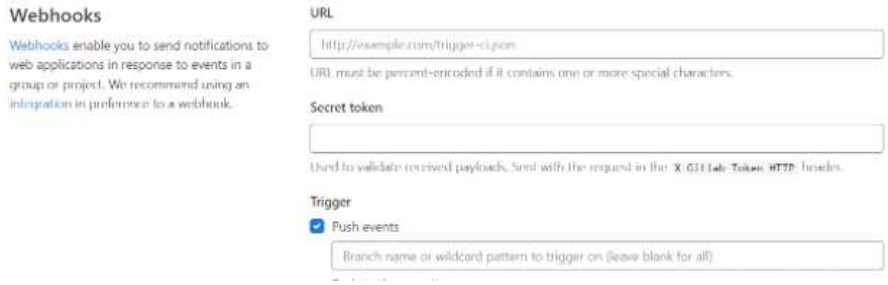
– Jenkins sudo 권한 부여

```
$ sudo nano /etc/sudoers
```

```
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
jenkins ALL=(ALL) NOPASSWD: ALL
```

해당 부분에 jenkins ALL=(ALL) NOPASSWO: ALL 작성

- Gitlab Webhook 작성
  - project settings - webhooks
    - url : 젠킨스 빌드 유발 url
    - secret token : 빌드 유발 고급에서 생성했던 시크릿 토큰
    - trigger : push events 체크 (위 설정했던 브랜치 입력)
    - add webhook



8. Frontend CI/CD

Backend와 CI/CD 방식 동일

Build - Execute Shell

```
echo "======jenkins build 프론트엔드 started...========"
pwd
cd frontend
docker stop reactapp
docker rm reactapp
docker build -t reactapp .
docker run -p 3000:3000 --name reactapp --network ubuntu_default -d reactapp
```

9. Django(Bigdata API) CI/CD

Backend와 CI/CD 방식 동일

Build - Execute Shell

```
echo "======jenkins build 빅데이터 started...========"
cd bigdata
chmod -R 777 ./manage.py

docker stop bigdata
docker rm bigdata
docker build -t bigdata .
docker run -p 7777:7777 --name bigdata --network ubuntu_default -d bigdata
```