# Jenkins 자동화 배포

# EC2 설정

1. 필요한 bash 명령어

* //현재 인스턴스 IP 출력  
  curl ident.me  
    
  //Port 방화벽 풀기  
  sudo ufw allow <Port>

# Docker, Docker-Compose 설치

## 우분투에 도커를 설치한다.

* 도커 설치 코드
* #오래된 버전의 도커 삭제  
  sudo apt-get remove docker docker-engine docker.io containerd runc  
    
  #repository 설정  
    
  ##apt package update  
  sudo apt-get update  
    
  ##Https를 통해 repository를 이용하기 위해 package들을 설치  
  sudo apt-get install \  
   ca-certificates \  
   curl \  
   gnupg \  
   lsb-release  
    
  ##Docker의 Official GPG Key 등록  
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg  
    
  ##stable repository를 등록  
  echo \  
   "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \  
   $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
    
  #Docker Engine 설치  
  sudo apt-get update  
  sudo apt-get install docker-ce docker-ce-cli containerd.io  
    
  #설치 확인  
  docker --version
* 도커 설정
* #사용자 권한 설정, 설정 후 재시작을 해야 적용이 된다.  
  sudo usermod -aG docker $USER  
    
  #컴포즈 설치  
  sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
  ##컴포즈 권한 설정  
  sudo chmod +x /usr/local/bin/docker-compose  
  ##컴포즈 설치 확인  
  docker-compose --version

## docker-compose.yml

version: "3.8"  
services:  
 jenkins:  
 privileged: true  
 build: /home/ubuntu/jenkins  
 container\_name: jenkins  
 restart: always  
 user: root  
 ports:  
 - "9090:8080"  
 - "50000:50000"  
 volumes:  
 - type: bind  
 source: /home/ubuntu/jenkins/home  
 target: /var/jenkins\_home  
 - type: bind  
 source: /var/run/docker.sock  
 target: /var/run/docker.sock  
 environment:  
 TZ: "Asia/Seoul"  
 tty: true  
 nginx:  
 container\_name: nginx  
 image: nginx:alpine  
 ports:  
 - "80:80"  
 - "443:443"  
 volumes:  
 - ./data/nginx:/etc/nginx/conf.d  
 - ./data/certbot/conf:/etc/letsencrypt  
 - ./data/certbot/www:/var/www/certbot  
 - ./nginx/var/www/dist:/var/www/dist  
 certbot:  
 container\_name: certbot  
 image: certbot/certbot  
 volumes:  
 - ./data/certbot/conf:/etc/letsencrypt  
 - ./data/certbot/www:/var/www/certbot  
networks:  
 default:  
 name: my\_net

1. 먼저 default 네트워크를 my\_net이라 잡는다. 이 네트워크로 컨테이너 간 통신을 한다.
2. jenkins의 home, docker.sock을 바인딩 시켜 젠킨스 컨테이너를 삭제해도 데이터는 남도록 한다.
3. jenkins는 8080포트를 받지만 9090으로 받도록 지정했다.
4. nginx의 포트 80, 443을 열어 http, https를 받을 수 있도록 한다.
5. nginx 컨테이너의 conf.d폴더, letsencrypt폴더, certbot폴더, dist폴더를 바인딩한다.
6. certbot 컨테이너의 letsencrypt폴더, certbot폴더를 바인딩한다. 이 때 nginx의 컨테이너의 letsencryt, certbot폴더와 같은 곳을 바인딩 시켜야 한다.

* **docker-compose up -d명령어로 실행**
  1. PermissionError: [Errno 13] Permission denied 오류 발생 시
     1. **sudo chmod 666 /var/run/docker.sock** 실행
  2. 도커 컨테이너는 \*\*docker exec -it [컨테이너 이름] bash\*\*로 실행한다.
     1. alpine 이미지를 가지면 \*\*docker exec -it [컨테이너 이름] /bin/sh\*\*로 실행한다.

# Git Lab 구조

## 구조도



## Dockerfile

### nuxt

FROM nginx  
RUN apt-get update  
RUN apt-get install npm -y  
  
# 웹서버 설정파일 복사  
# 프로젝트 루트 밑에 docker폴더, default.conf 생성  
COPY ./frontend/letcipe/nginx/default.conf /etc/nginx/conf.d/default.conf  
WORKDIR /var/www/dist  
  
COPY ./frontend/letcipe/package\*.json ./  
# 모듈 설치  
RUN npm install  
# 소스 복사  
COPY ./frontend/letcipe ./  
  
# 빌드  
RUN npm run build  
# 컨테이너 실행시 시작  
CMD [ "npm", "run", "start" ]

### back

FROM openjdk:11-jdk  
  
#포트번호 설정  
EXPOSE 8888  
ENV TZ=Asia/Seoul  
#가져올 파일 jar또는 war  
ARG JAR\_FILE=./backend/build/libs/\*.jar  
  
#war파일 복사  
COPY ${JAR\_FILE} app.jar  
  
#ENTRYPOINT 명령을 지정, app.jar 실행  
ENTRYPOINT ["java","-jar","/app.jar"]

## docker-compose.yml

version: "3.8"  
services:  
 nuxt:  
 depends\_on:  
 - deployback  
 build:  
 context: ./  
 dockerfile: ./frontend/letcipe/Dockerfile  
 network: my\_net  
 container\_name: nuxt  
 user: root  
 ports:   
 - 4000:4000  
 - 3000:3000  
   
 deployback:  
 depends\_on:  
 - mysql  
 build:   
 context: ./  
 dockerfile: ./backend/Dockerfile  
 network: my\_net  
 container\_name: back  
 user: root  
 volumes:  
 - back:/var/letcipe  
 environment:  
 SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql:3306/ssafy  
 SPRING\_DATASOURCE\_USERNAME: root  
 SPRING\_DATASOURCE\_PASSWORD: fPttlvl0914  
 ports:  
 - 8888:8888  
 mysql:  
 container\_name: mysql  
 user: root  
 image: mysql:8.0.28 # 컨테이너에서 사용하는 base image 지정  
 restart: always # 컨테이너 다운 시 재시작하라는 명령어  
 volumes: # -v 옵션과 동일  
 - mysql:/var/lib/mysql  
 environment: # 컨테이너 안의 환경변수 설정  
 - MYSQL\_ROOT\_PASSWORD=fPttlvl0914  
 - MYSQL\_DATABASE=ssafy  
 ports: # -p 옵션과 동일  
 - "3306:3306"  
volumes:  
 mysql:  
 name: mysql  
 back:  
 name: back  
networks:  
 default:  
 external:  
 name: my\_net

1. 외부 네트워크(EC2 도커에 있는 docker-compose.yml의 default network)인 my\_net을 가져와 default로 잡는다.
2. depends\_on 조건에 따라 mysql이 가장 먼저 설치가 되고 mysql의 볼륨을 따로 잡아준 후 환경 설정을 해준다.
3. 백엔드에서는 백엔드의 볼륨을 잡아주고 환경 설정을 통해 mysql을 연동시켜준다.
4. 백엔드와 프론트엔드는 docker hub에서 갖고 오는 것이 아닌 Dockerfile을 통해 docker image가 만들어지므로 build 조건에 맞춰서 이미지가 생성되고 docker-compose up 명령을 통해 컨테이너가 생성이 된다.
5. nuxt는 기본적으로 3000포트를 사용한다.

# Nginx

## app.conf

server {  
 listen 80;  
 listen [::]:80;  
 server\_name j7a705.q.ssafy.io;  
 location / {  
 return 301 https://$host$request\_uri;  
 }   
}  
upstream back {  
 server back:8888;  
}  
upstream nuxt {  
 server nuxt:3000;  
}  
  
server {  
 listen 443 ssl;  
 listen [::]:443 ssl;  
 server\_name j7a705.q.ssafy.io;  
 # server\_name \_;  
 ssl\_certificate /etc/letsencrypt/live/j7a705.q.ssafy.io/fullchain.pem;  
 ssl\_certificate\_key /etc/letsencrypt/live/j7a705.q.ssafy.io/privkey.pem;  
  
 location / {  
 proxy\_connect\_timeout 600s;  
 proxy\_read\_timeout 600s;  
 proxy\_send\_timeout 600s;  
 proxy\_buffers 8 16k;  
 proxy\_buffer\_size 32k;  
 proxy\_pass http://nuxt;  
 # root /var/www/dist;  
 }  
 location /api {  
 proxy\_pass http://back/api;  
 }  
 include /etc/letsencrypt/options-ssl-nginx.conf;  
 ssl\_dhparam /etc/letsencrypt/ssl-dhparams.pem;  
}

1. 80포트로 요청이 오면 443포트로 리다이렉션 시킨다.
2. upstream으로 서버 내부 도커 네트워크로 소통을 한다.
   1. 백엔드 : 8888
   2. 프론트엔드 : 3000 (SSR)
3. url이 ‘/’로 들어오면 프론트엔드 컨테이너로 요청을 보낸다.
4. url이 ‘/api’로 들어오면 백엔드 컨테이너로 요청을 보낸다.

# SSL 인증서 발급

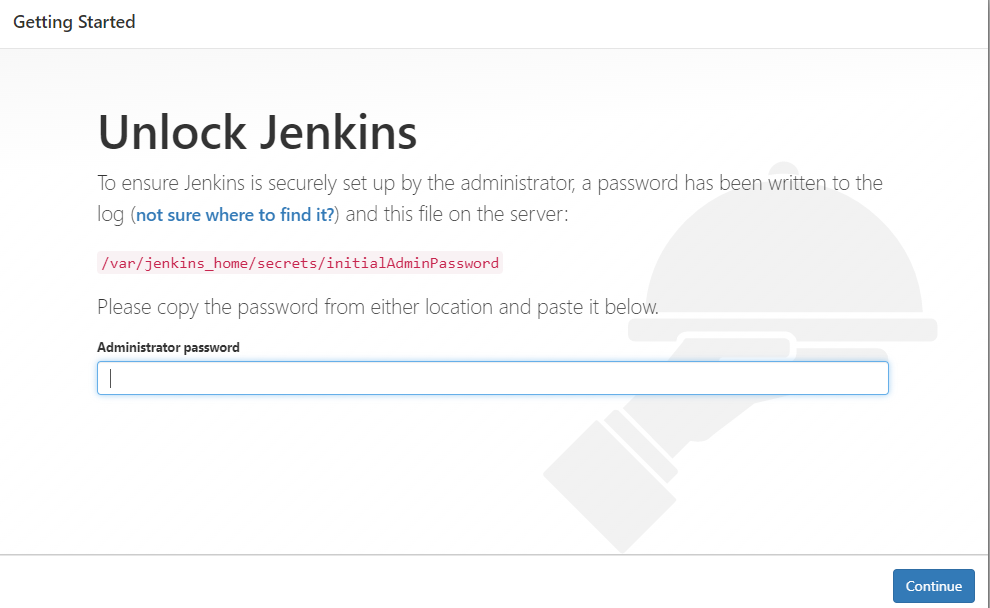
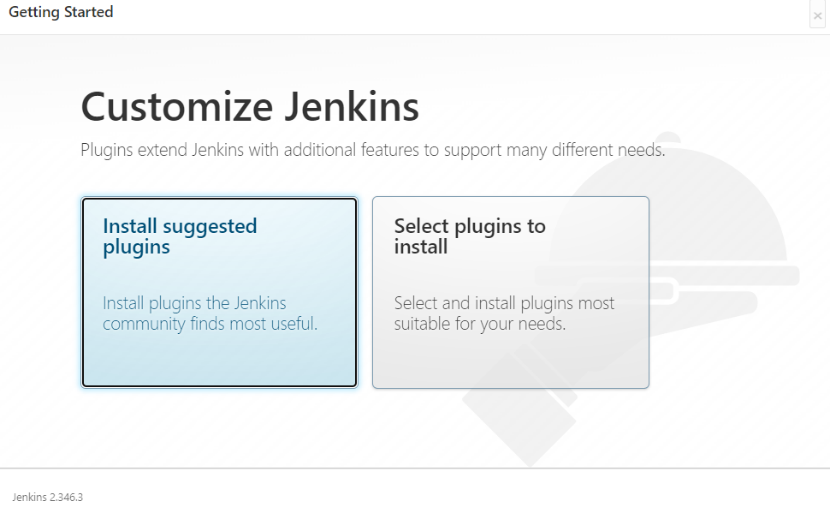
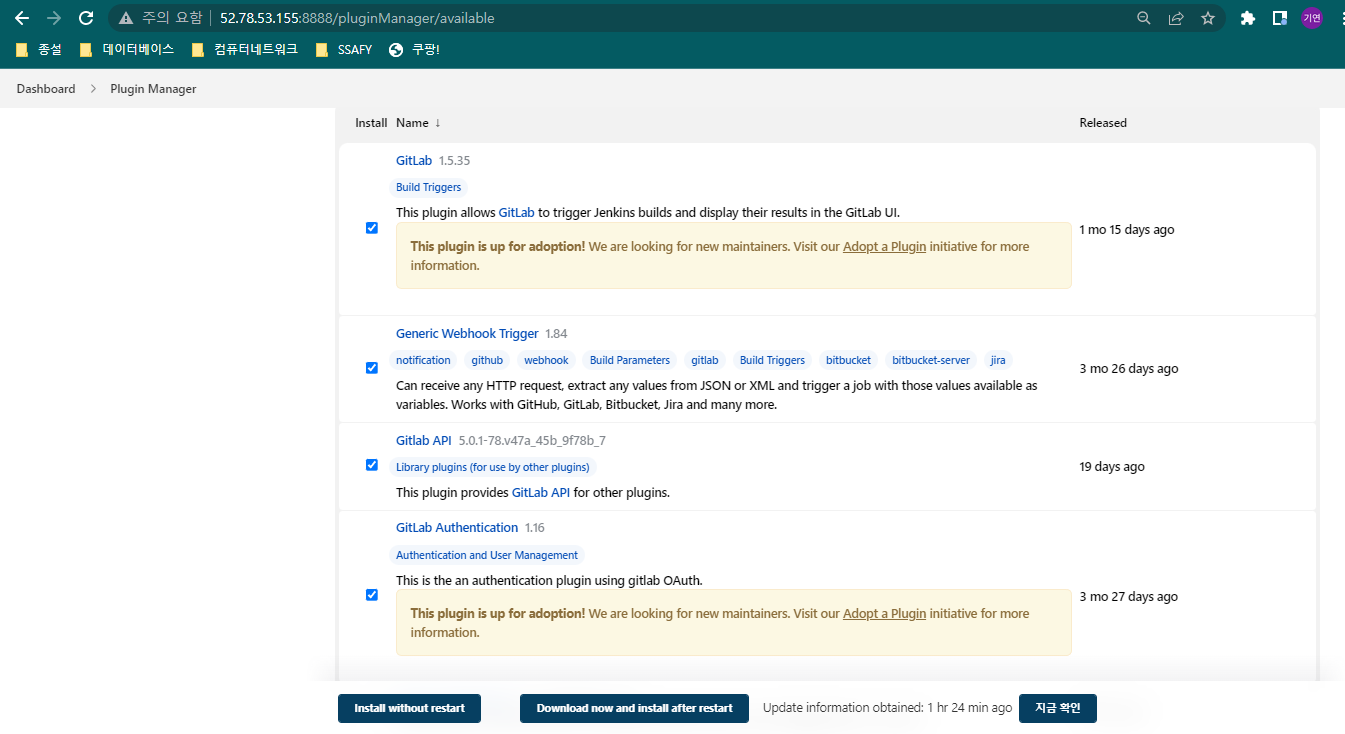
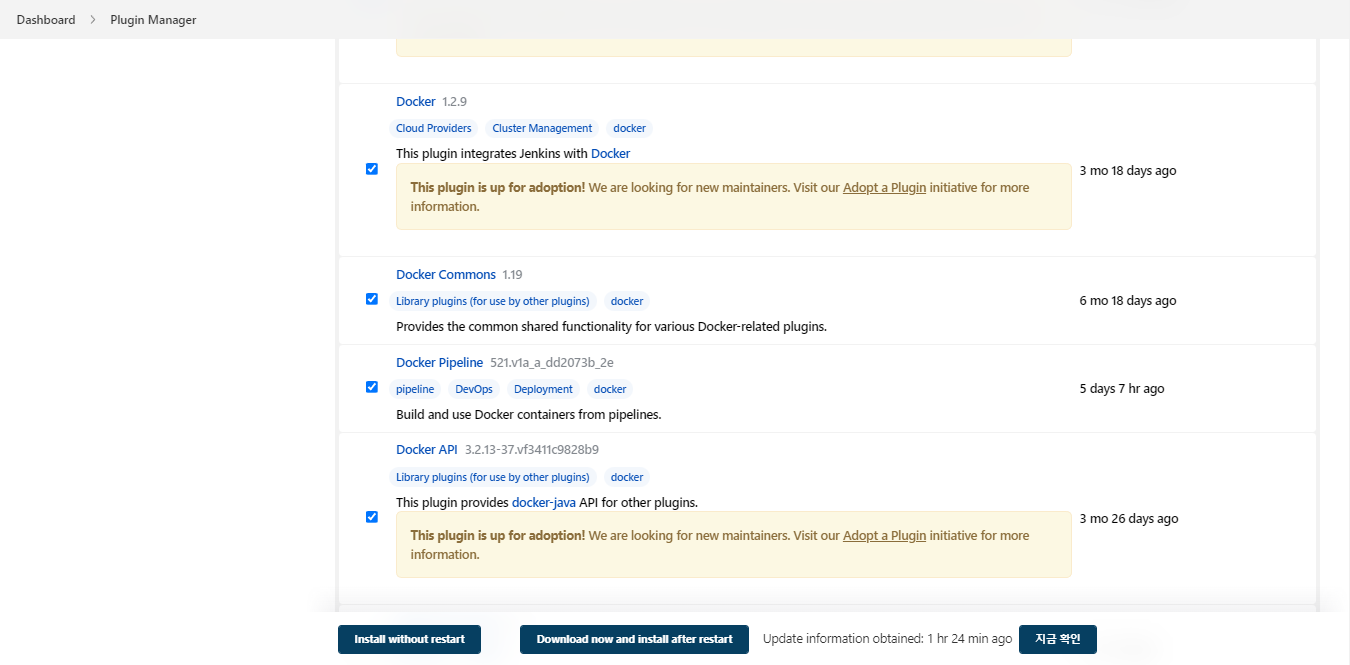
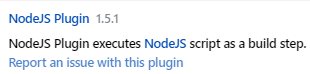
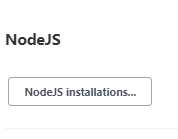
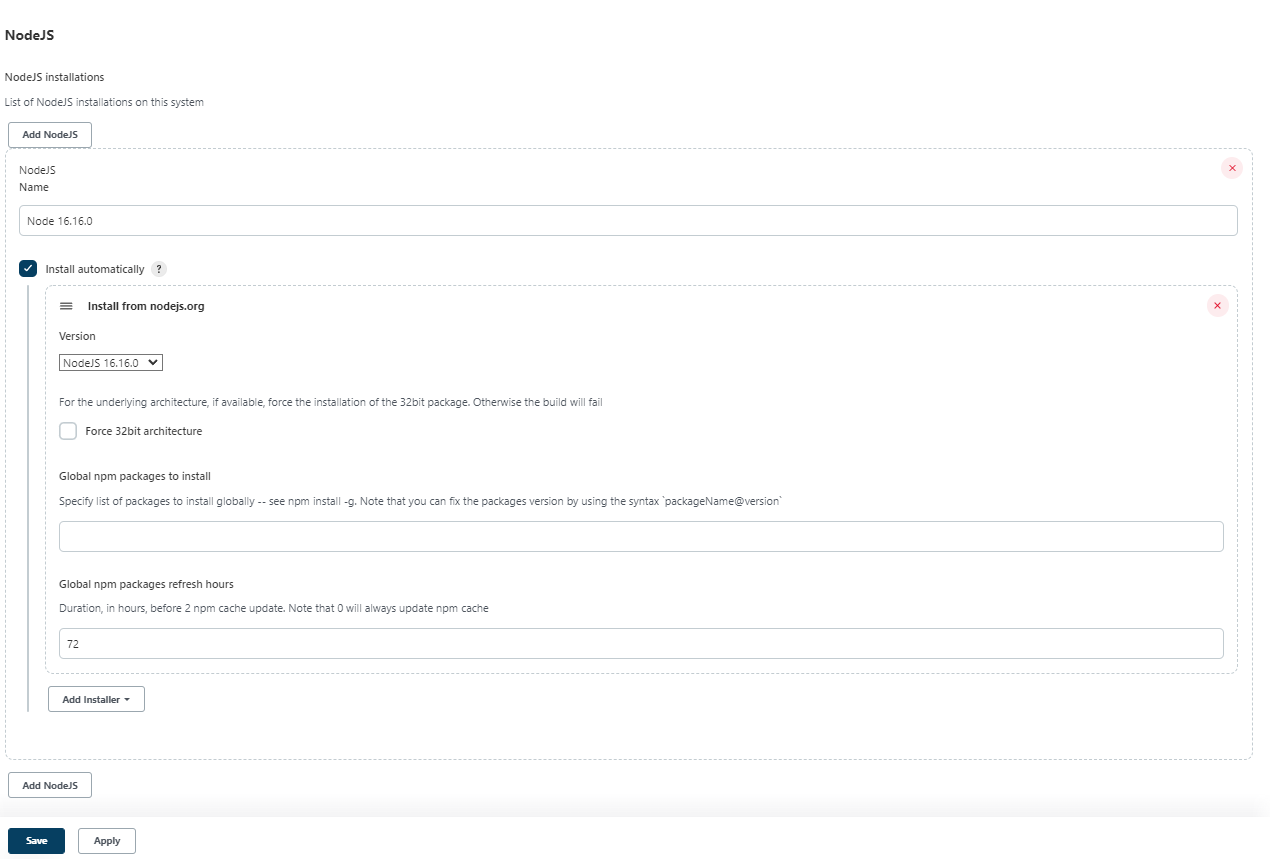
## init-letsencrypt.sh

#!/bin/bash  
  
if ! [ -x "$(command -v docker-compose)" ]; then  
 echo 'Error: docker-compose is not installed.' >&2  
 exit 1  
fi  
  
domains=(j7a705.q.ssafy.io)  
rsa\_key\_size=4096  
data\_path="./data/certbot"  
email="giyeon3145@gmail.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 "$data\_path/conf"  
 curl -s https://raw.githubusercontent.com/certbot/certbot/master/certbot-nginx/certbot\_nginx/\_internal/tls\_configs/options-ssl-nginx.conf > "$data\_path/conf/options-ssl-nginx.conf"  
 curl -s https://raw.githubusercontent.com/certbot/certbot/master/certbot/certbot/ssl-dhparams.pem > "$data\_path/conf/ssl-dhparams.pem"  
 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 nginx  
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 --webroot -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 nginx nginx -s reload

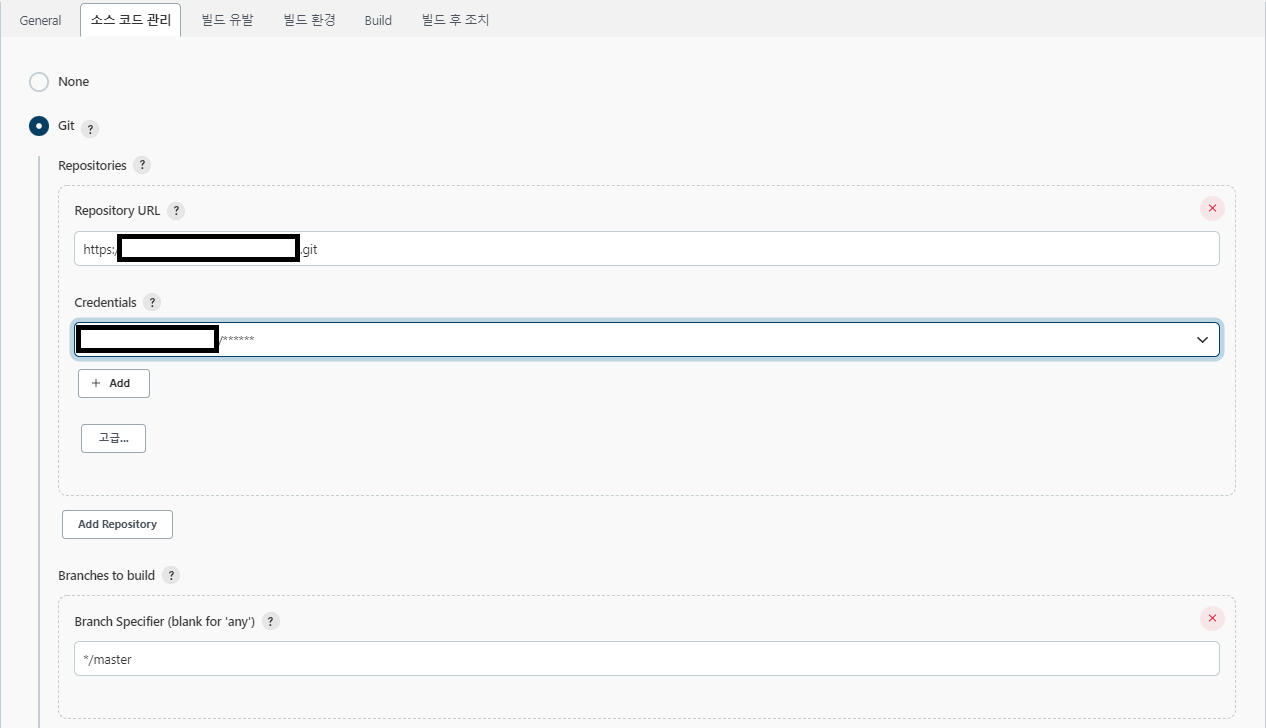
1. domains에 도메인 이름을 작성한다.
2. data/path에 바인딩 시킨 폴더명을 작성한다.
3. email를 작성한다.
4. staging을 0으로 작성하면 ssl 인증서를 발급 받고 1로 지정하면 test용으로 발급된다.
5. 만약 ssl 인증서를 발급 받았다면 docker-compose.yml에서 certbot부분을 주석처리해준다.

# 젠킨스

## 젠킨스 설정

1. **도커 컴포즈로 설치한 jenkins 컨테이너를 실행시킨다.**
2. **docker-in-docker 형식으로 도커 설치**
   * 명령어
   * #Docker 설치  
     ##오래된 버전의 도커 삭제  
     apt-get remove docker docker-engine docker.io containerd runc  
     ##Repository 설정  
     apt-get update  
     apt-get install \  
      ca-certificates \  
      curl \  
      gnupg \  
      lsb-release  
     mkdir -p /etc/apt/keyrings  
       
     ##키 등록  
     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  
     ##Docker Engine 설치  
     apt-get update  
     apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
3. **도커 설치가 끝나면 지정해준 포트 번호로 들어가서 다음과 같은 작업을 해준다.**
   1. 젠킨스 접속
   2. 터미널에서 cat /var/jenkins\_home/secrets/initialAdminPassword 입력 후 나온 코드 입력 후 continue
   3. install suggested plugins
   4. 플러그인 설치
      1. gitlab, generic webhook trigger, gitlab api, gitlab authentication
      2. docker, docker commons, docker pipeline, docker api
      3. Nodejs
   5. jenkins관리 → global tool configuration
      1. NodeJS → NodeJS installations…
      2. 이름, 버전 맞추기
      3. save 눌러서 저장

## 젠킨스 아이템 생성

1. **아이템 이름(letcipe) 작성 후 Freestyle project**
2. **소스 코드 관리에 URL, Credentials 작성 및 브랜치 설정**
   1. URL : gitlab URL
   2. Credentials → add jenkins
      1. Username : gitlab 아이디
      2. password : gitlab 비밀번호
3. **빌드 유발에서 “Build when a change is pushed to GitLab. GitLab webhook URL:** [**http:///project/letcipe**](http://52.78.53.155:8888/project/firstDeploy) **Enabled GitLab triggers” 체크**
4. **빌드 환경에서 “Delete workspace before build starts”를 체크 후 아래와 같이 조건을 추가한다.**
5. **빌드 환경에서 Provide Node & npm bin/forder to PATH 체크**
6. **Build**

* docker cp ./hadoop/functions/loggenerator/\* datanode:/home/letcipe/data  
  for file in ./hadoop/functions/mapreduce/\*;   
  do   
   docker cp $file datanode:/hadoop/dfs/data;   
  done  
  docker exec -i datanode bash /hadoop/dfs/data/make\_jar.sh  
  docker-compose down  
  docker-compose up -d
  1. LogGenerator, 하둡 java파일들을 hadoop datanode 컨테이너 안에 복사해준다.
  2. datanode의 make\_jar.sh을 실행시켜준다.
  3. git lab에 올라와 있는 docker-compose.yml 파일을 바탕으로 컨테이너를 만들어준다.

1. **save**
2. **해당 item에서 지금 빌드 실행**

# Hadoop(Big Data Europe)

## repository 클론

\*\*git clone https://github.com/big-data-europe/docker-hadoop-spark-workbench.git\*\*

## hadoop container 띄우기

\*\*docker-compose -f docker-compose-hive.yml up -d\*\*

* **docker-compose-hive.yml**
* version: '2'  
  services:  
   namenode:  
   image: bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8  
   user: root  
   container\_name: namenode  
   volumes:  
   - ./data/namenode:/hadoop/dfs/name  
   - back:/home/letcipe/data  
   environment:  
   - CLUSTER\_NAME=test  
   env\_file:  
   - ./hadoop-hive.env  
   ports:  
   - 50070:50070  
   datanode:  
   image: bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8  
   user: root  
   container\_name: datanode  
   environment:  
   - TZ=Asia/Seoul  
   depends\_on:  
   - namenode  
   volumes:  
   - ./data/datanode:/hadoop/dfs/data  
   - back:/home/letcipe/data  
   env\_file:  
   - ./hadoop-hive.env  
   tty: true  
   ports:  
   - 9999:9999  
   - 50075:50075  
  volumes:  
   back:  
   external: true  
  networks:  
   default:  
   external:  
   name: my\_net
  1. 스프링 부트와 같은 볼륨을 잡기 위해 docker-compose-hive.yml에 back 볼륨을 외부에서 가져온다.

## shell script

\*\*shell script 실행하기 전 chmod +x [shell\_script\_name].sh 명령을 입력해야 한다.\*\*

* **jar 파일 생성 스크립트(make\_jar.sh)**
* REPOS=$(ls -d /hadoop/dfs/data/\*)  
  for repo in $REPOS  
  do  
   echo $repo  
   name=${repo:17}  
   echo $name  
   if [[ "$repo" == \*java\* ]]; then  
   name=${name:0:(-5)}  
   echo $name  
   $(mkdir /hadoop/dfs/data/repo\_${name})&  
   wait  
   $(mv /hadoop/dfs/data/${name}.java /hadoop/dfs/data/repo\_${name})&  
   fi  
  done  
    
  REPOS=$(ls -d /hadoop/dfs/data/\*)  
  for repo in $REPOS  
  do  
   if [[ "$repo" == \*repo\_\* ]]; then  
   echo $repo  
   name=${repo:17}  
   jar\_name=${repo:22}  
   echo $name  
   cd $repo  
   echo $JAVA\_HOME  
   export PATH=${JAVA\_HOME}/bin:${PATH}  
   export HADOOP\_CLASSPATH=${JAVA\_HOME}/lib/tools.jar  
   wait  
   $(hadoop com.sun.tools.javac.Main \*.java)&  
   wait  
   $(jar cf ${jar\_name}.jar \*.class)&  
   cd $repo  
   cd ..  
   fi  
  done
  1. jenkins 생성 시 만들어지는 java파일들이 있으면 “repo\_”를 앞에 붙여 폴더를 만들어 주고 java파일을 만든 폴더 안에 넣어준다.
  2. 폴더를 다 만든 다음 이름을 갖고 오고 만든 폴더 안에 들어간다.
  3. 경로를 만든 폴더 위치로 잡고 하둡 경로도 잡아준다.
  4. 하둡 명령어로 java파일을 컴파일해준다.
  5. 그 다음 jar 명령어로 컴파일한 파일들을 jar파일로 압축해서 만들어 준다.
* **하둡 실행 스크립트(exec\_[java\_file\_name].sh)**
* name=$(date -d 'yesterday' "+%Y-%m-%d")  
    
  echo "$name"  
  wait  
  $(hdfs dfs -mkdir -p /\*\*[java\_file\_name]\*\*/$name)&  
  wait  
  $(hdfs dfs -put /home/letcipe/data/log/\*\*[java\_file\_name]\*\*/$name/\* /\*\*[java\_file\_name]\*\*/$name)&  
  wait  
  $(hadoop jar \*\*[java\_file\_name]\*\*.jar \*\*[java\_file\_name]\*\* /\*\*[java\_file\_name]\*\*/$name /output)&  
  wait  
  $(mkdir -p /home/letcipe/data/\*\*[java\_file\_name]\*\*\_result)&  
  wait  
  $(rm /home/letcipe/data/\*\*[java\_file\_name]\*\*\_result/${name}.txt)  
  wait  
  $(hdfs dfs -get /output/part-r-\* /home/letcipe/data/\*\*[java\_file\_name]\*\*\_result/${name}.txt)&  
  wait  
  $(hdfs dfs -rm -r /output)&  
  wait  
  $(hdfs dfs -rm -r /\*\*[java\_file\_name]\*\*)&
  1. 이름 변수에 어제 날짜로 주고 hdfs에 폴더를 만들어준다.
  2. hdfs에 만든 폴더에 데이터를 넣어준다
  3. jar파일을 이용해 하둡을 실행한다.
  4. back 볼륨이 잡혀있는 위치에 결과 파일을 저장할 폴더를 생성한다.
  5. 만약 저장할 위치에 이름 변수와 동일한 파일이 있으면 삭제해준다
  6. 실행한 결과를 저장 위치에 복사해준다
  7. hdfs에 있는 결과 폴더와 넣어준 데이터들을 지워준다.

## Log Generator

* LogGenerator.java
* import java.io.BufferedWriter;  
  import java.io.File;  
  import java.io.FileWriter;  
  import java.io.IOException;  
  import java.time.LocalDate;  
  import java.util.ArrayList;  
  import java.util.List;  
  import java.util.Random;  
    
  /\*\*  
   \* Hello world!  
   \*/  
  public class LogGenerator {  
   static int now = LocalDate.now().getYear();  
    
   public static void main(String[] args) throws IOException {  
   addLog(args[0], Integer.parseInt(args[1]));  
   }  
    
   public enum JOB {  
   STUDENT, JUBU, WORKER, COOK  
   }  
    
   public static void addLog(String base, int loopCount) throws IOException {  
   // loop  
   Random rnd = new Random();  
   // 파일 디렉토리 생성  
   // 지난 한달 사이의 랜덤한 날짜로 뽑기  
   for (int day = 10; day < 30; day++) {  
   String date = String.format("2022-09-%d", day);  
    
   String path = base + date;  
   for (int i = 0; i < 10; i++) {  
   // 파일 생성  
   File dir = new File(path);  
   if (!dir.exists())  
   dir.mkdirs();  
   File file = new File(path + String.format("/cartlog.%s.%d.log", date, i));  
   BufferedWriter bw = new BufferedWriter(new FileWriter(file));  
    
   for (int l = 0; l < loopCount; l++) {  
   // 랜덤하게 로그 한줄씩 추가  
   StringBuilder sb = new StringBuilder();  
   int h = rnd.nextInt(14) + 10;  
   int m = rnd.nextInt(50) + 10;  
   int s = rnd.nextInt(50) + 10;  
    
   String time = String.format("%d:%d:%d", h, m, s);  
   int rid = recipeList[rnd.nextInt(recipeList.length)];  
   String category = "R" + rnd.nextInt(15);  
   int uid = rnd.nextInt(1000);  
   String gender = rnd.nextBoolean() ? "F" : "M";  
   String birth = String.format("199%d-0%d-22", rnd.nextInt(9) + 1, rnd.nextInt(9) + 1);  
   JOB job = JOB.values()[rnd.nextInt(4)];  
   int family = rnd.nextInt(4) + 1;  
   String log = String.format("{RECIPE:{id:%d,category:%s},USER:{gender:%s,nickname:admin,birth:%s,id:%d,job:%s,family:%d}}", rid, category, gender,  
   birth, uid, job.toString(), family);  
   // 파일 저장  
   sb.append(String.format("[INFO ] [%s %s] [%s]", date, time, log));  
   bw.write(sb.toString());  
   bw.write("\n");  
   // loop end  
   }  
   bw.flush();  
   bw.close();  
   }  
   }  
   }  
    
   public void combinationTest() {  
   System.out.println("Hello World!");  
   String value = "[INFO ] [2022-09-28 13:27:46] [{RECIPE:{id:26,category:R0001},USER:{gender:M,nickname:admin,birth:1997-03-17,id:1,job:WORKER,family:4}}]";  
   List<String> tokens = logParser(value);  
   StringBuilder result = new StringBuilder();  
   int flag = 0;  
   while (++flag < 16) {  
   // 플래그 분석하여 key 생성  
   StringBuilder sb = new StringBuilder();  
   for (int i = 0; i < 4; i++) {  
   if ((flag & (1 << i)) != 0) {  
   sb.append(tokens.get(i));  
   } else {  
   sb.append("-");  
   }  
   sb.append(",");  
   }  
   sb.setLength(sb.length() - 1);  
   result.append(sb.toString()).append("\n");  
   }  
   System.out.println(result.toString());  
   }  
    
   static List<String> logParser(String log) {  
   String[] parseSB = log.split("\\[");  
   List<String> list = new ArrayList<>();  
   String[] parseCB = parseSB[3].split("\\{");  
   if (parseCB[1].contains("}")) {  
   parseCB[1] = parseCB[1].substring(0, parseCB[1].indexOf('}'));  
   }  
   String[] parseCL;  
   parseCL = parseCB[3].split("\\:|\\}");  
   for (int i = 1; i <= 5; i++) {  
   if (i == 2 || i == 4)  
   continue;  
   if (parseCL[i].contains(",")) {  
   parseCL[i] = parseCL[i].substring(0, parseCL[i].indexOf(','));  
   } else {  
   parseCL[i] = parseCL[i].substring(0, parseCL[i].indexOf('}'));  
   }  
   list.add(parseCL[i]);  
   }  
   String[] date = list.get(1).split("\\-");  
   list.remove(1);  
   list.add(1, "" + ((now - Integer.parseInt(date[0])) / 10 \* 10));  
   list.add(2, parseCL[6]);  
   parseCL = parseCB[2].split("\\:");  
   parseCL[1] = parseCL[1].substring(0, parseCL[1].indexOf(','));  
   list.add(parseCL[1]);  
    
   return list;  
   }  
    
   static final int[] recipeList = {  
   1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 19, 21, 23, 25, 26, 27, 28, 30, 31, 32, 33, 34, 37, 38, 41, 42, 43, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 67, 68, 69, 70, 72, 73, 74, 75, 76, 78, 79, 80, 81, 82, 83, 84, 85, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 112, 113, 114, 115, 118, 119, 120, 121, 122, 125, 127, 128, 129, 130, 131, 133, 134, 135, 137, 138, 139, 140, 141, 142, 144, 145, 146, 147, 148, 149, 150, 152, 153, 156, 157, 158, 160, 161, 162, 163, 164, 165, 166, 167, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 184, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 210, 211, 212, 213, 215, 216, 217, 218, 220, 221, 223, 224, 225, 226, 227, 228, 230, 231, 234, 236, 237, 238, 239, 240, 242, 243, 244, 245, 246, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 269, 270, 271, 272, 273, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 293, 294, 296, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 314, 315, 316, 317, 318, 319, 320, 322, 324, 325, 326, 327, 328, 329, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 343, 344, 345, 346, 347, 348, 349, 351, 352, 353, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 368, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 437, 438, 439, 440, 441, 442, 443, 444, 446, 447, 448, 449, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 463, 464, 465, 466, 467, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 490, 492, 493, 496, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 510, 511, 514, 515, 516, 517, 519, 522, 523, 524, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 582, 583, 587, 588, 589, 590, 591, 594, 595, 596, 598, 599, 600, 606, 607, 608, 609, 610, 611, 614, 615, 617, 618, 620, 621, 622, 624, 625, 630, 631, 632, 634, 636, 637, 638, 639, 640, 641, 642, 643, 644, 647, 649, 651, 652, 653, 654, 655, 656, 658, 659, 660, 661, 663, 664, 665, 666, 667, 669, 670, 671, 672, 674, 675, 676, 677, 679, 681, 685, 686, 687, 692, 693, 694, 696, 697, 698, 699, 700, 701, 702, 704, 705, 706, 708, 709, 710, 711, 712, 713, 716, 717, 718, 719, 720, 721, 723, 724, 726, 728, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 745, 746, 749, 751, 752, 755, 756, 759, 760, 761, 762, 764, 769, 771, 772, 773, 775, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 789, 790, 794, 795, 798, 799, 802, 803, 805, 806, 807, 808, 810, 814, 815, 816, 817, 820, 821, 822, 828, 831, 832, 835, 838, 839, 844, 845, 848, 853, 856, 859, 861, 863, 868, 869, 872, 874, 876, 877, 882, 886, 894, 898, 900, 901, 902, 903, 905, 909, 910, 916, 917, 924, 926, 929, 930, 931, 934, 939, 940, 941, 945, 948, 949, 950, 952, 954, 957, 963, 965, 967, 968, 970, 989, 990, 997, 998, 999, 1000, 1002, 1003, 1004, 1005, 1006, 1009, 1010, 1011, 1012, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1024, 1025, 1026, 1028, 1030, 1031, 1032, 1033, 1035, 1036, 1038, 1039, 1040, 1041, 1042, 1043, 1045, 1046, 1048, 1049, 1050, 1053, 1054, 1057, 1060, 1061, 1062, 1063, 1064, 1065, 1069, 1071, 1072, 1073, 1074, 1075, 1077, 1081, 1084, 1085, 1086, 1088, 1089, 1090, 1091, 1092, 1093, 1096, 1097, 1098, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1131, 1133, 1137, 1138, 1139, 1140, 1145, 1147, 1149, 1150, 1151, 1153, 1156, 1157, 1158, 1159, 1163, 1166, 1168, 1169, 1171, 1174, 1175, 1176, 1177, 1181, 1183, 1184, 1185, 1188, 1190, 1192, 1196, 1198, 1199, 1201, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1215, 1217, 1222, 1223, 1224, 1239, 1241, 1242, 1257, 1262, 1265, 1266, 1267, 1269, 1280, 1284, 1285, 1288, 1289, 1290, 1293, 1298, 1302, 1303, 1305, 1306, 1309, 1317, 1318, 1326, 1327, 1328, 1329, 1357, 1358, 1359, 1360, 1361, 1362, 1364, 1365, 1368, 1369, 1371, 1372, 1374, 1378, 1386, 1388, 1389, 1390, 1391, 1394, 1398, 1400, 1401, 1402, 1404, 1405, 1406, 1407, 1410, 1415, 1418, 1420, 1421, 1423, 1425, 1427, 1428, 1429, 1431, 1432, 1438, 1439, 1440, 1441, 1444, 1446, 1447, 1448, 1449, 1452, 1453, 1454, 1455, 1457, 1462, 1467, 1468, 1471, 1477, 1478, 1480, 1482, 1488, 1491, 1492, 1493, 1494, 1495, 1497, 1498, 1499, 1500, 1501, 1502, 1503, 1504, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 1516, 1517, 1518, 1520, 1521, 1522, 1523, 1525, 1526, 1527, 1528, 1529, 1530, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1541, 1547, 1550, 1551, 1552, 1554, 1563, 1565, 1568, 1569, 1570, 1574, 1579, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1590, 1592, 1593, 1594, 1607, 1611, 1612, 1614, 1616, 1618, 1631, 1634, 1635, 1637, 1641, 1642, 1643, 1644, 1646, 1647, 1649, 1650, 1651, 1652, 1654, 1655, 1656, 1657, 1659, 1660, 1661, 1662, 1665, 1667, 1672, 1673, 1675, 1676, 1677, 1678, 1680, 1681, 1682, 1684, 1685, 1688, 1689, 1693, 1694, 1695, 1697, 1701, 1704, 1709, 1710, 1712, 1714, 1715, 1716, 1718, 1719, 1730, 1732, 1734, 1735, 1736, 1741, 1743, 1747, 1749, 1755, 1768, 1773, 1777, 1780 };  
  }

1. CartCount에 해당하는 로그를 생성하는 자바 파일이다.
2. \*\*javac LogGenerator.java\*\* 로 컴파일을 하고 \*\*java LogGenerator [저장할 위치] [Amount] 로 실행한다.\*\*
   1. **로그 양=Amount\**20\*10***

# 배포 특징

## SSR(Server Side Rendering)

1. nuxt.js는 배포를 할 때 2가지 방법이 있다. 첫 번째로 정적 파일을 이용한 배포, 두 번째로는 SSR을 이용한 배포이다.
   1. SSR : Server Side Rendering, 서버로부터 완전하게 만들어진 html파일을 받아와 페이지 전체를 렌더링 하는 방식
2. SSR에는 장단점이 분명한데 장점으로 검색엔진 최적화(SEO)가 있어 채택을 했다. 단점으로는 서버 렌더링에 따른 부하가 심해 성능이 낮아진다는 단점이 있다. 향후 검색 엔진 기능을 강화할 때 유용하게 쓰일 것 같다.
   1. SEO : Search Engine Optimization

## Hadoop, Spring Boot Volume 연결

1. 하둡과 스프링 부트의 볼륨을 연결하여 같은 디렉토리를 사용할 수 있도록 했다.
2. 하둡에서 처리한 결과 파일을 Volume으로 지정한 곳에 저장을 하여 Spring Boot에서 통신을 따로 하지 않아도 바로 가져와 사용할 수 있도록 만들었다.