

나날

포팅 매뉴얼

구미 1반 10조 dbd

박소연, 박소희, 이동욱, 임유정, 임지원, 하진우

목차

I. 개요.....	- 3 -
1. 프로젝트 개요.....	- 3 -
2. 주요 기술	- 3 -
3. 사용한 외부 서비스 목록.....	- 4 -
II. 프론트 엔드 빌드.....	- 5 -
1. Git clone.....	- 5 -
2. ignore된 파일 추가.....	- 5 -
3. Dockerfile, nginx.conf 파일 작성.....	- 5 -
4. 빌드.....	- 6 -
5. 배포.....	- 6 -
III. 백엔드 빌드	- 7 -
1. Git clone.....	- 7 -
2. ignore 된 파일 추가	- 7 -
3. 카카오 API 관련 설정.....	- 10 -
4. 구글 앱 비밀번호 발급	- 10 -
5. Dockerfile 작성.....	- 11 -
6. 빌드.....	- 11 -
7. 배포.....	- 12 -
IV. Python Flask.....	- 12 -
1. 파이썬 가상 환경 만들기.....	- 12 -

2. 플라스크 프로젝트 생성하기	- 13 -
3. 파이참에서 myproject open & 인터프리터 위치 설정 & 디버깅 설정	- 14 -
4. requirements.txt 파일 생성	- 14 -
5. Dockerfile 만들기	- 14 -
6. Docker build & push	- 15 -
V. 서버 세팅	- 15 -
1. EC2 세팅	- 15 -
2. EC2 서버에 docker 설치	- 16 -
3. EC2 서버에 MariaDB 설치	- 17 -
4. DB bash 접속, 계정 생성	- 17 -
5. AWS S3 Bucket 설정	- 20 -
6. Nginx Default 값	- 23 -
VI. 자동 배포 : Jenkins	- 25 -
1. Jenkins 플러그인 설치	- 25 -
2. Jenkins 프로젝트 생성	- 26 -
3. Frontend 배포	- 26 -
4. Backend 배포	- 30 -

I. 개요

1. 프로젝트 개요

친구와 손쉽게 나의 일상을 공유할 수는 없을까? 나의 일상을 좀 더 진솔하게 기록할 수는 없을까? 나의 일상을 좀 더 특별히 기록할 수 있는 방법은 무엇일까?

‘나날’은 나날이를 의미하는 단어입니다.

일기를 나날이 써보자, 나의 일상을 나날이 좀 더 특별히 기록해보자, 친구와 나의 나날을 공유해보자와 같은 의미를 지니고 있습니다.

‘나날’은 일기를 기록하고 그림을 만들어주며 친구와 공유할 수 있는 웹 서비스입니다.

AI를 통하여 일기의 내용을 그림으로 그려주기도 하며, 그날의 감정을 분석하여 사용자에게 알려줌으로서 나의 일상을 좀 더 특별히 기록이 가능합니다. 또한, 이러한 일기들을 친구들과 손쉽게 공유하는 장을 제공함으로써 교환 일기의 플랫폼을 제공하고 있습니다.

2. 주요 기술

Backend - Spring

IntelliJ IDE

Springboot Gradle 6.8.3

Java jdk corretto 11.0.17

Spring Data JPA

Spring Security 5.6.1

Spring Validation 2.5.6

Spring Web

Frontend - React

Visual Studio Code IDE

Nodejs 18.12.1

React 18.2.0

TailwindCss

sweetalert2 11.3.10

axios 1.2.3

Swagger 2.7.7

Lombok

Backend - DB

MariaDB 10.10.2

Amazon S3

Backend - Flask

python 3.9

java-1.7-openjdk

Scikit-learn 1.0.2

openai 0.26.5

nltk 3.8.1

pandas 1.5.3

konlpy

JPyype1==1.4.1

CI/CD

AWS EC2 Ubuntu 20.04 Docker

20.10.23

Jenkins

NGINX

SSL

3. 사용한 외부 서비스 목록

- 네이버 파파고 API
- Dalle API
- 카카오 로그인 API
- 구글 메일 API

II. 프론트 엔드 빌드

1. Git clone

1. Clone 시 Master로 Clone 하기

```
git clone https://lab.ssafy.com/s08-webmobile2-sub2/S08P12D110.git
```

2. Clone 받은 폴더로 이동

```
cd S08P12D110
```

3. Branch를 FE/develop으로 변경

```
git checkout --track origin/FE/develop
```

2. ignore된 파일 추가

.env.local

```
REACT_APP_API_BASE_URL=http://[도메인]/naanal/
```

해당 변수 사용법 : process.env.REACT_APP_API_BASE_URL

3. Dockerfile, nginx.conf 파일 작성

Dockerfile

```
FROM nginx:stable-alpine  
WORKDIR /app  
RUN mkdir ./build  
ADD ./build ./build  
RUN rm /etc/nginx/conf.d/default.conf
```

```
COPY ./nginx.conf /etc/nginx/conf.d
```

```
EXPOSE 80
```

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

nginx.conf

```
server {  
    listen 80;  
    location / {  
        root /app/build;  
        index index.html;  
        try_files $uri $uri/ /index.html;  
    }  
}
```

4. 빌드

프로젝트 폴더에 있는 front_Nanal 디렉토리의 루트 경로에서 다음과 같은 명령어를 입력합니다

```
npm i
```

```
npm run build
```

5. 배포

```
docker build -t [이미지명]:[태그명] .
```

```
docker run --name [컨테이너명] -d -p 3000:80 [이미지명]:[태그명]
```

III. 백엔드 빌드

1. Git clone

1. Clone 시 Master로 Clone 하기

```
git clone https://lab.ssafy.com/s08-webmobile2-sub2/S08P12D110.git
```

2. Clone 받은 폴더로 이동

```
cd S08P12D110
```

3. Branch를 BE/develop으로 변경

```
git checkout -track origin/BE/develop
```

2. ignore 된 파일 추가

아래의 파일들을 src/main/resources 폴더 안에 생성합니다.

application-aws.properties

: AWS EC2의 데이터베이스 정보와, AWS S3 관련 내용을 담고 있습니다.

```
application-aws.properties

spring.datasource.driverClassName=org.mariadb.jdbc.Driver

spring.datasource.url=jdbc:mysql://{서버주소}:{포트번호}/{DB이름}?serverTimezone=Asia/Seoul&useUnicode=true&characterEncoding=utf8

spring.datasource.username={DB username}

spring.datasource.password={DB password}
```



```
# s3 설정
```

```
cloud.aws.credentials.accessKey={AWS accessKey}
```

```
cloud.aws.credentials.secretKey={AWS secretKey}
```

```
cloud.aws.s3.bucket=nanal-dbd
```

```
cloud.aws.region.static=ap-northeast-2
```

```
cloud.aws.stack.auto=false
```

```
application-oauth.properties
```

: 카카오 소셜 로그인에 필요한 설정들을 담고 있습니다. 카카오 API 설정 관련 내용은 별도 내용을 확인해주세요.

```
spring.security.oauth2.client.registration.kakao.client-id={KAKAO_REST_API_KEY}
```

```
spring.security.oauth2.client.registration.kakao.client-name=Kakao
```

```
spring.security.oauth2.client.registration.kakao.client-secret={KAKAO_SECRET_KEY}
```

```
spring.security.oauth2.client.registration.kakao.scope=profile_nickname, profile_image, account_email, friends
```

```
spring.security.oauth2.client.registration.kakao.redirect-uri=http://{도메인}/nanal/login/oauth2/code/kakao
```

```
spring.security.oauth2.client.registration.kakao.client-authentication-method=POST
```

```
spring.security.oauth2.client.registration.kakao.authorization-grant-type=authorization_code
```

```
# provider
```

```
spring.security.oauth2.client.provider.kakao.authorization-  
uri=https://kauth.kakao.com/oauth/authorize  
  
spring.security.oauth2.client.provider.kakao.token-  
uri=https://kauth.kakao.com/oauth/token  
  
spring.security.oauth2.client.provider.kakao.user-info-  
uri=https://kapi.kakao.com/v2/user/me  
  
spring.security.oauth2.client.provider.kakao.user-name-attribute=id
```

application-email.properties

: 이메일 발송에 필요한 설정들을 담고 있습니다.

```
mail.smtp.auth=true  
  
mail.smtp.starttls.required=true  
  
mail.smtp.starttls.enable=true  
  
mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory  
  
mail.smtp.socketFactory.fallback=false  
  
mail.smtp.port=465  
  
mail.smtp.socketFactory.port=465  
  
  
AdminMail.id ={구글 이메일 주소}  
AdminMail.password ={앱 비밀번호}
```

application-naver.properties

: 일기 그림 생성에 이용하는 네이버, Dalle API 정보를 담고 있습니다.

```
Client-ID={네이버 Client ID}  
  
Client-Secret={네이버 Client SecretKey}
```

```
Dalle-API-Key={Dalle API Key}
```

application-jwt.properties

: JWT 비밀키를 담고 있습니다. 아래와 같은 내용을 입력해주세요.

```
secretKey = {64Byte 이상의 문자열}
```

3. 카카오 API 관련 설정

1. Kakao Developers(<https://developers.kakao.com/>) 접속 및 로그인
2. [내 애플리케이션] → [애플리케이션 추가하기]
3. 'REST API 키' 복사, *application-oauth.properties* 파일
{KAKAO_REST_API_KEY} 위치에 추가
4. [앱설정] → [플랫폼] → Web 플랫폼 등록 → 사이트 도메인 추가

```
https://{도메인}
```

5. Redirect URI 등록

```
https://{도메인}/naanal/login/oauth2/code/kakao
```

6. [앱설정] → [동의항목] → 닉네임, 프로필 사진, 카카오회정(이메일), 카카오회스 내 친구목록 사용
7. [앱설정] → [보안] → Client Secret 코드 생성
8. 발급받은 Secret 코드를 *application-oauth.properties* 파일
{KAKAO_SECRET_KEY} 위치에 추가

4. 구글 앱 비밀번호 발급

1. 구글(<https://google.com/>) 접속 및 로그인

이 때, 로그인에 사용한 이메일을 *application-email.properties* 파일 {구글 이메일 주소}에 입력

2. 프로필 이미지 클릭 → [Google 계정 관리] 선택
3. [보안] → [Google에 로그인] → [앱 비밀번호] 선택
4. 앱 : 메일, 기기 : Windows 컴퓨터 선택 후 생성
5. 생성된 비밀번호를 복사한 후 *application-email.properties* 파일 {앱 비밀번호}에 입력

5. Dockerfile 작성

Dockerfile

```
# base-image # 항상 베이스 이미지로 시작해야 한다.  
FROM openjdk:11-jdk  
  
# 변수설정 (빌드파일의 경로)  
ARG JAR_FILE=build/libs/*.jar  
  
# 파일복사 : 자주 안바뀌는 파일부터 COPY하기  
# 빌드파일을 컨테이너로 복사  
COPY ${JAR_FILE} app.jar  
  
# jar 파일 실행  
ENTRYPOINT ["java","-jar","/app.jar"]
```

6. 빌드

1. GUI 이용시 (IntelliJ IDEA 2022.3.1 Ultimate Edition 기준)

- 1) *Gradle* 선택
- 2) *nanal/Tasks/build/Clean* 더블 클릭
- 3) *nanal/Tasks/build/BootJar* 더블 클릭

2. Command 사용시

프로젝트 폴더 내에 있는 *back_Nanal* 디렉토리의 루트 경로에서 다음의 명령어를 실행합니다.

```
gradle clean build
```

7. 배포

```
docker build -t [이미지명]:[태그명] .
```

```
docker run --name [컨테이너명] -d -p 8080:8080 [이미지명]:[태그명]
```

IV. Python Flask

1. 파이썬 가상 환경 만들기

```
# 가상환경 만들기

'경로'>mkdir venvs

cd '경로' venvs

python -m venv myproject # 가상 환경 진입

cd '경로'\venvs\myproject\Scripts

activate

# 가상 환경에서 플라스크 설치하기
```

```
pip install flask

# pip 최신 버전으로 설치하기
python -m pip install --upgrade pip
```

2. 플라스크 프로젝트 생성하기

```
# 프로젝트 디렉토리 생성하기
cd '경로'

# 프로젝트들의 루트 디렉토리 만들기
mkdir projects

# 가상환경 진입
cd '경로'\projects
'경로'\venvs\myproject\Scripts\activate

# 플라스크 프로젝트를 담을 myproject 디렉토리 생성
mkdir myproject
cd myproject
```

※ 환경변수 에러 - 'JAVA_HOME' 경로 적어주기

```
import os

os.environ['JAVA_HOME'] = r'C:\Program Files\Zulu\zulu-8'
```

3. 파이참에서 myproject open & 인터프리터 위치 설정 & 디버깅 설정

setting > project: myproject > project interpreter > project interpreter 톱니설정 > existing environment ... > scripts - python.exe

```
# 파일 이름이 app이 아닐 경우  
set FLASK_APP=[이름]  
set FLASK_DEBUG=true  
flask run
```

4. requirements.txt 파일 생성

requirements.txt

```
pip freeze > requirements.txt
```

```
// 현재 환경에 설치된 python package와 version 정보 저장
```

5. Dockerfile 만들기

Dockerfile

```
FROM python:3.9  
  
# install java  
ENV JAVA_HOME /usr/lib/jvm/java-1.7-openjdk/jre  
RUN apt-get update && apt-get install -y g++ default-jdk  
RUN pip install konlpy
```

```
WORKDIR /app
```

```
# 해당 디렉토리에 있는 모든 하위항목들을 '/app'으로 복사
```

```
COPY . /app
```

```
RUN pip install -r requirements.txt
```

```
CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

6. Docker build & push

```
docker build -f Dockerfile -t [도커이미지이름] .
```

```
docker push [도커이미지이름]
```

V. 서버 세팅

1. EC2 세팅

1. TimeZone 설정

```
SET GLOBAL time_zone='ASIA/SEOUL';
```

```
SET time_zone='+09:00';
```

```
flush privileges;
```

```
date
```


2. EC2 서버에 docker 설치

1. apt를 이용하여 docker를 설치할 예정이라 apt를 update 합니다.

```
sudo apt update
```

2. docker 설치에 필요한 패키지들을 설치합니다.

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

3. curl을 이용, 도커를 설치하기 위한 내용을 다운로드 받고, apt 기능을 위한 리스트에 추가합니다.

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

4. ubuntu 18.04 버전에 맞는 docker를 다운로드 할 수 있도록 repository 리스트에 추가합니다.

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
```

5. apt update를 실행합니다.

```
sudo apt update
```

→ apt list에 도커를 다운로드 할 경로가 업데이트 되었습니다.

6. docker-ce를 설치합니다.(커뮤니티 버전)

```
apt-cache policy docker-ce
```

```
sudo apt install docker-ce
```

7. docker가 설치되면, 자동으로 시스템 서비스로서 등록이 됩니다.

systemctl 명령어를 통해 docker 서비스 상태를 확인해 보면, 도커엔진이 구동중인 상태 확인

```
sudo systemctl status docker
```

3. EC2 서버에 MariaDB 설치

1. MariaDB 설치

```
sudo docker pull mariadb
```

2. 3306 포트로 실행

```
sudo docker run -p 3306:3306 --name mariadb -e  
MARIADB_ROOT_PASSWORD=[비밀번호] -d mariadb
```

- *-name*: my-mariadb 라는 컨테이너 이름을 부여
- *-p 3306:3306*: host port number:container port number
- *-e*: -e는 환경 변수 옵션이다.
-e MARIADB_ROOT_PASSWORD=비밀번호: 비밀번호를 지정
- *-d*: detached 모드에서 실행되어 백그라운드로 실행한다.
- *mariadb*: 앞서 받은 이미지 이름

4. DB bash 접속, 계정 생성

1. bash 접속

```
sudo docker exec -it mariadb bash
```

2. DB 버전확인

```
mysql --version
```

3. DB 접속

```
mysql -u root -p
```

4. 사용자 추가 : localhost에서만 접속 가능한 계정 생성

```
# mysql>
```

```
use mysql;
```

```
# CREATE USER 'YOUR_SYSTEM_USER'@'localhost' IDENTIFIED BY
'YOUR_PASSWD';

create user 'localhost'@'localhost' identified by '비밀번호';

# GRANT ALL PRIVILEGES ON *.* TO 'YOUR_SYSTEM_USER'@'localhost';

grant all privileges on *.* to 'localhost'@'localhost';

flush privileges;
```

5. 모든 DB, 테이블에 접속 가능한 계정 생성

```
USE mysql;

CREATE USER 'localhost'@'%' IDENTIFIED BY 'd110';

GRANT ALL PRIVILEGES ON *.* TO 'localhost'@'%';

FLUSH PRIVILEGES;

USE mysql;

CREATE USER 'user'@'%' IDENTIFIED BY 'd110';

GRANT ALL PRIVILEGES ON *.* TO 'user'@'%';

FLUSH PRIVILEGES;
```

6. 데이터베이스 생성

```
create database [데이터베이스명]
```

7. SSH접속과 외부 접속 허용

```
vi /etc/mysql/mariadb.conf.d/50-server.cnf

# bind-address = 0.0.0.0
```

```
# 추가해주기 :wq 로 저장 후 나가기
```

8. MariaDB charset 변경

```
# apt 업데이트
```

```
apt-get update
```

```
# vim 설치
```

```
apt-get install vim
```

```
vi /etc/mysql/my.cnf
```

9. *my.cnf*에 아래 내용 추가

```
[client]
```

```
default-character-set=utf8mb4
```

```
[mysql]
```

```
default-character-set=utf8mb4
```

```
[mysqld]
```

```
character-set-server=utf8mb4
```

```
collation-server=utf8mb4_unicode_ci
```

```
skip-character-set-client-handshake
```

10. Hostname public IP로 변경

```
curl http://169.254.169.254/latest/meta-data/public-ipv4
```

5. AWS S3 Bucket 설정

1. config, access

build.gradle

```
implementation 'org.springframework.cloud:spring-cloud-starter-aws:2.2.6.RELEASE'
```

2. Key value

Application-aws.properties 파일에 아래 내용이 포함되어 있는지 확인해주세요.

application-aws.properties

```
cloud.aws.credentials.accessKey= {AWS accessKey}
cloud.aws.credentials.secretKey= {AWS secretKey}
cloud.aws.s3.bucket=arn:aws:s3:::[버킷이름]
cloud.aws.region.static=[pacific..]
cloud.aws.stack.auto=false
```

3. aws policy (버킷-권한-버킷 정책)

policy generator 활용 버킷 정책 설정

(<http://awspolicygen.s3.amazonaws.com/policygen.html>)

s3 bucket policy, *, get object&put object, *arn:aws:s3:::버킷이름*(ARN)

Select Policy Type

container for permissions. The different types of policies you can create are an [IAM Policy](#), an [S3 Bucket Policy](#), an [IAM Role Policy](#), and an [SQS Queue Policy](#).

Select Type of Policy S3 Bucket Policy ▼

Add Statement(s)

is the formal description of a single permission. See [a description of elements](#) that you can use in statements.

Effect ☒ Allow ☐ Deny

Principal

Use a comma to separate multiple values.

AWS Service Amazon S3 ▼

☐ All Services ('*')

Use multiple statements to add permissions for more than one service.

Actions 2 Action(s) Selected ▼ ☐ All Actions ('*')

Resource Name (ARN)

ARN should follow the following format: arn:aws:s3:::\${BucketName}/\${KeyName}.
Use a comma to separate multiple values.

[Add Conditions \(Optional\)](#)

[Add Statement](#)

```
{
  "Id": "Policy1675521463879",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1675521426271",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::[ARN오류]/*",
```

```

    "Principal": "*"
  }
]
}

```

// 모든 대상(Principal)에 대해서, nanal-dbd 버킷에 있는 모든 파일 (Resource)에 대한, 객체 가져오기, 객체 업로드 하는 특정 행동 (Action)을 허용(Effect) 한다는 뜻이다.

4. CORS 설정(버킷-권한-Cross-origin 리소스 공유)

```

[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "HEAD",
      "GET",
      "PUT",
      "POST",
      "DELETE"
    ],
    "AllowedOrigins": [
      "http://localhost:3000",
      "https://[ip주소]:[포트번호]"
    ]
  }
]

```

```

    ],
    "ExposeHeaders": [
        "x-amz-server-side-encryption",
        "x-amz-request-id",
        "x-amz-id-2",
        "Access-Control-Allow-Origin"
    ],
    "MaxAgeSeconds": 3000
}
]

```

6. Nginx Default 값

1. 서버 Default 값 설정

Server

```

server {
    # 프론트 연결

    location /{

        proxy_pass http://localhost:3000;

    }


    # 백엔드 연결

    location /nana1 {

        proxy_pass http://localhost:8080/nana1;
    }
}

```



```

    }

    listen 443 ssl; # 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
}

server {

    if ($host = {도메인}) {

        return 301 https://$host$request_uri;

    } # managed by Certbot


    listen 80;

    server_name {도메인};

    return 404; # managed by Certbot

}

```

2. Nginx 실행

```
sudo systemctl start nginx
```

3. Front Nginx

```
server {  
    listen 80;  
    location / {  
        root /app/build;  
        index index.html;  
        try_files $uri $uri/ /index.html;  
    }  
}
```

VI. 자동 배포 : Jenkins

1. Jenkins 플러그인 설치

Jenkins 관리 → 플러그인 관리 → Available plugins

1. SSH

- Publish Over SSH
- SSH Agent Plugin

2. GitLab

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

3. Docker

- Docker API Plugin
- Docker Commons Plugin
- Docker Pipeline
- Docker plugin

4. NodeJS

- NodeJS Plugin

2. Jenkins 프로젝트 생성

1. 젠킨스 메인 페이지 → 새로운 Item → Pipeline
2. Jenkins 관리 → Credentials → add credentials
 - gitlab
 - Docker hub
 - Ssh
3. webhook 설정

3. Frontend 배포

Pipeline Script

```
pipeline {
    environment{
        registry="[user name]/[repository]"
        registryCredential='dockerHub-access-token'
        dockerImage=' '
    }
}
```

```

agent any

stages {
    stage('git clone') {
        steps {
            git branch: 'FE/develop',
                credentialsId: 'access-token',
                url: 'https://lab.ssafy.com/s08-webmobile2-
sub2/S08P12D110'
        }
    }
    stage('build'){
        steps{
            sh "cp /var/jenkins_home/property/.env.local
/var/jenkins_home/workspace/Front/front_Nanal"
            dir('front_Nanal'){
                sh'''
                    npm install
                    CI='' npm run build
                '''
            }
        }
    }
    stage('build docker'){
        steps{

```

```

        dir('front_Nanal'){
            sh "pwd"
            sh "ls -al"
            script{
                dockerImage=docker.build registry
            }
        }
    }
}

stage('clean image'){
    steps{
        sshagent(credentials:['ssh-access-token']){
            sh '''
                ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker stop nanal_front "
                ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker rm nanal_front"
                ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker rmi [user name]/[repository]:front1.0"
            '''
        }
    }
}

stage('push docker'){

```

```

        steps{
            script {
                docker.withRegistry('', registryCredential){
                    dockerImage.push("front1.0")
                }
            }
        }
    }
}

stage('ssh-server'){
    steps{
        sshagent(credentials:['ssh-access-token']){
            sh '''

                ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker pull [user name]/[repository]:front1.0"

                ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker run --name nanal_front -d -p 3000:80
[user name]/[repository]:front1.0"

            '''
        }
    }
}
}
}

```

4. Backend 배포

Pipeline Script

```
pipeline {
    environment{
        registry="[user name]/[repository]"
        registryCredential='dockerHub-access-token'
        dockerImage=' '
    }
    agent any
    stages {
        stage('git clone') {
            steps {
                git branch: 'BE/develop',
                    credentialsId: 'access-token',
                    url: 'https://lab.ssafy.com/s08-webmobile2-
sub2/S08P12D110'
            }
        }
        stage('build'){
            steps{
                sh "cp /var/jenkins_home/property/*.properties
/var/jenkins_home/workspace/Back/back_Nanal/src/main/resources"
                dir('back_Nanal'){
                    sh'''
```

```

        pwd

        echo build start

        chmod +x ./gradlew

        ./gradlew clean bootJar

        '''

    }

}

stage('build docker'){
    steps{
        dir('back_Nanal'){
            sh "pwd"

            sh "ls -al"

            script{
                dockerImage=docker.build registry
            }
        }
    }
}

stage('clean image'){
    steps{
        sshagent(credentials:['ssh-access-token']){
            sh '''

```



```

        ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker stop nanal_back "

        ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker rm nanal_back"

        ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker rmi [user name]/[repository]:1.0"

        '''

    }

}

stage('push docker'){
    steps{
        script {
            docker.withRegistry('', registryCredential){
                dockerImage.push("1.0")
            }
        }
    }
}

stage('ssh-server'){
    steps{
        sshagent(credentials:['ssh-access-token']){
            sh '''

                ssh -o StrictHostKeyChecking=no

```

```
ubuntu@3.36.51.212 "docker pull [user name]/[repository]:1.0"

        ssh -o StrictHostKeyChecking=no
ubuntu@3.36.51.212 "docker run --name nanal_back -d -p 8080:8080
[user name]/[repository]:1.0"

        ...

        }

    }

}

}
```