



바꾸바꾸

A708 : 바꾸바꾸

삼성 청년 SW 아카데미 서울캠퍼스 8 기

공통 프로젝트

23.01.03 ~ 23.02.17

포팅 매뉴얼

담당 컨설턴트 : 김성준

이안채(팀장), 김소정, 배정현, 이승희, 이은지, 최웅렬

목차

1. 프로젝트 기술 스택.....	3
2. 서버 세팅.....	5
3. 빌드 상세내용	15
4. 외부 서비스.....	26

1. 프로젝트 기술 스택

1. 이슈관리: Jira
2. 형상관리: Git
3. 커뮤니케이션: Mattermost, Webex, notion
4. 개발환경
 - OS: Window 10
 - IDE
 - IntelliJ
 - VSCode
 - Figma
 - Database
 - DBMS: Mariadb 10
 - NoSQL: Mongodb 5.0.14
 - Cache: Redis 7.0.7
 - SearchEngine: ElasticSearch
 - Server: AWS EC2
 - OS: Ubuntu 20.04 LTS (GNU/Linux 5.4.0-1018-aws x86_64)
 - File Server: AWS S3
 - CI/CD: Jenkins, Docker, Nginx
5. 상세 기술
 - Frontend
 - React.js : 17
 - React Query
 - TailwindCSS
 - Styled-Component
 - Zustand
 - Backend

- JDK: 11
- Spring Boot: 2.7.7
- Gradle
- Spring Security
- Spring Data JPA
- Springfox Swagger UI: 2.9.2
- Lombok
- Logger
- Json Web Token
- AWS
- Spring data mongodb reactive
- Spring Webflux
- Data-redis
- Server
 - AWS EC2
 - AWS S3
 - AWS CloudFront
 - Ubuntu 20.04 LTS
 - Docker
 - Jenkins
 - CertBot
- IDE
 - HeidiSQL
 - IntelliJ IDEA
 - VSCode
 - VIM

2. 서버 세팅

2-1. Docker 설치

서버에서 도커를 설치해야 각종 작업을 수행할 수 있다.

아래의 명령어를 EC2 서버에 접속해서 순차적으로 실행한다.

```
apt-get update

apt-get install sudo
apt-get install vim
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
sudo add-apt-repository \W
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \W
$(lsb_release -cs) stable"
sudo apt-get update && sudo apt-get install docker-ce docker-ce-cli containerd.io
```

이후, 도커 컴포즈도 설치해야 프로젝트를 도커 컴포즈를 활용하여 빌드할 수 있다.

```
sudo apt install jq

VERSION=$(curl --silent https://api.github.com/repos/docker/compose/releases/latest | jq .name -r)
DESTINATION=/usr/bin/docker-compose
sudo curl -L https://github.com/docker/compose/releases/download/${VERSION}/docker-compose-
$(uname -s)-$(uname -m) -o $DESTINATION
sudo chmod 755 $DESTINATION
```

도커 컨테이너끼리 네트워크로 통신을 하기 위해서는 도커 네트워크를 생성해야 하므로 아래의 명령어를 통해 네트워크를 생성한다.

```
sudo docker network create cd_network
```

2-2. HTTPS 설정

도메인은 가비아를 통해서 미리 DNS 설정을 끝마쳤다는 가정 하에 진행한다.

1) 22, 80, 443, 8080 포트 개방

```
sudo ufw allow 22/tcp  
  
sudo ufw allow 80/tcp  
sudo ufw allow 443/tcp  
sudo ufw allow 8080/tcp  
sudo ufw enable
```

2) Certbot 설치

```
sudo apt-get update  
  
sudo snap install core; sudo snap refresh core  
sudo snap install certbot --classic
```

3) Nginx 설치

```
sudo apt install nginx
```

4) nginx 설정

```
cd /etc/nginx/conf.d  
  
sudo vim default.conf
```

jenkins 의 nginx 설정은 아래의 링크를 참조하여 진행한다.

<https://www.jenkins.io/doc/book/system-administration/reverse-proxy-configuration-nginx/>

```

server {
    server_name <서버 주소>;

    proxy_set_header Host $host;
    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;

    location /api/ {
        rewrite ^/api/(.*)$ /$1 break;
        proxy_pass http://localhost:9999;
    }

    location /chatapi/ {
        rewrite ^/chatapi/(.*)$ /$1 break;
        proxy_pass http://localhost:9997;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    location /notifyapi/ {
        rewrite ^/notifyapi/(.*)$ /$1 break;
        proxy_pass http://localhost:9998;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

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

```

```

server {
    if ($host = <서버 주소>) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;
    server_name baggu.shop;
    return 404; # managed by Certbot
}

```


5) Certbot nginx 에 적용

```
sudo certbot --nginx
```

이 과정을 거치게 되면 nginx 의 default.conf 에 자동으로 ssl 인증서가 적용된다.

6) nginx 시작

```
sudo nginx
```

추후에 nginx 설정이 변경되면 아래와 같은 명령어들을 이용한다.

```
sudo nginx -t  
sudo service nginx restart
```

2-3. Jenkins 설정

CI / CD 구축을 위해 jenkins 를 사용하려면 아래의 과정을 따른다.

젠킨스 이미지 다운로드

```
sudo docker pull jenkins/jenkins:lts-jdk11
```

jenkins 를 실행하기 위해서는 아래의 명령어를 입력한다.

```
docker run -u 0 -d -p 9090:8080 -p 50000:50000 -v /var/jenkins:/var/jenkins_home -v  
/var/run/docker.sock:/var/run/docker.sock --name jenkins jenkins/jenkins:lts-jdk11
```

이후, sudo docker logs jenkins 명령어를 통해 jenkins 의 암호를 확인한다.

Nginx 로 라우팅처리한 url 을 통해 jenkins 에 접속한다.

Dashboard > Jenkins 관리 > Plugin Manager 로 접속한 뒤 Available plugins 에서 필요한 플러그인을 설치한다.

Gitlab 플러그인

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

Docker 플러그인

- Docker API Plugin
- Docker Commons Plugin
- Docker Compose Build Step Plugin
- Docker Pipeline
- Docker plugin
- docker-build-step

이 후 EC2 에서 아래의 명령어를 통해 Jenkins 컨테이너의 bash 로 접근한다.

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

jenkins 내부에서 docker 에 접근할 수 있도록 (도커 인 도커) 아래의 명령어를 실행한다.

```
apt-get update

apt-get install \
    ca-certificates \
    curl \
    gnupg \
    sb-release
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
apt-get update
apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

docker-compose 를 사용하기 위해 아래 명령어를 추가로 입력한다.

```
sudo apt install jq
VERSION=$(curl --silent https://api.github.com/repos/docker/compose/releases/latest | jq .name -r)
DESTINATION=/usr/bin/docker-compose
sudo curl -L https://github.com/docker/compose/releases/download/${VERSION}/docker-compose-
$(uname -s)-$(uname -m) -o $DESTINATION
sudo chmod 755 $DESTINATION
```

도커 인 도커의 준비가 끝났다면, Jenkins 웹훅을 설정한다. 자세한 내용은 아래의 블로그를 참조해서 작업한다.

<https://chhanz.github.io/devops/2020/05/04/jenkins-ci/>

젠킨스 파이프라인을 위해 사용된 코드는 아래와 같다.

```

pipeline {
  agent any
  environment {
    SKIP_BUILD = "${env.gitlabBranch != 'main'}"
  }
  stages {
    stage("Print Event Info") {
      when {
        expression { return SKIP_BUILD == "true" }
      }
      steps {
        echo "Branch : ${env.gitlabBranch}"
        echo "URL : ${env.gitlabSourceRepoURL}"
        echo "User : ${env.gitlabUserName}"
        echo "Action : ${env.gitlabActionType}"
      }
    }
    stage('Clone') {
      when {
        expression { return SKIP_BUILD == "true" }
      }
      steps {
        git branch: "${env.gitlabBranch}", url:
'https://lab.ssafty.com/s08-webmobile2-sub2/S08P12A708',
credentialsId: 'jenkins_webhook'
      }
    }
    stage('Docker compose down') {
      when {
        expression { return SKIP_BUILD == "true" }
      }
      steps {
        echo "docker compose down"
        sh "sudo docker-compose -f ci-docker-compose.yml
down --rmi all"
      }
    }
    stage('Docker compose build') {
      when {
        expression { return SKIP_BUILD == "true"}
      }
      steps {
        echo "docker compose build"
        sh "sudo docker-compose -f ci-docker-compose.yml
build --no-cache"
      }
      post {
        success {
          echo 'Success to build'
        }
        failure {
          echo 'Docker build failed. remove all test
images'
          sh "sudo docker-compose -f ci-docker-
compose.yml down --rmi all"
          error 'pipeline aborted.'
        }
      }
    }
    stage('Docker compose up') {
      when {
        expression { return SKIP_BUILD == "true" }
      }
      steps {
        echo "docker compose up"
        sh "sudo docker-compose -f ci-docker-compose.yml
up -d"
      }
      post {
        success {
          echo 'Success to run docker compose'
        }
        failure {
          echo 'Docker run failed. remove all test
images'
          sh "sudo docker-compose -f ci-docker-
compose.yml down --rmi all"
          error 'pipeline aborted.'
        }
      }
    }
    // stage('docker compose down') {
    //   when {
    //     expression { return SKIP_BUILD == "true" }
    //   }
    //   steps {
    //     echo "docker compose down"
    //     sh "sudo docker-compose -f ci-docker-
compose.yml down --rmi all"
    //   }
    // }
  }
}

```

이 젠킨스 파일을 통해 자동 build 를 수행하기 위해서는 Deploy 환경과 Test 환경의 분리가 필요하다. 자세한 내용은 3. 빌드 상세 내용에서 추가적으로 설명한다.

2-4. Mariadb 설정

Maria db 의 경우 EC2 서버 내부에 직접 접근하여 실행 시켜둔다. 아래는 이를 위한 도커 파일이다.

Dockerfile (~/database/Dockerfile)

```
# mariadb:10을 이미지로 사용한다.
FROM mariadb:10

# config 파일 카피
COPY ./config /etc/mysql/conf.d

# db 이름을 설정
ENV MARIADB_DATABASE=<사용할 db 이름>

# root 계정의 패스워드 설정
ENV MARIADB_ROOT_PASSWORD=<db 패스워드>

# 볼륨 마운트
VOLUME /var/lib/mysql

# Expose 포트 설정
EXPOSE 3306
```

mariadb.cnf (~/database/config/mariadb.cnf)

```
[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

[mysqldump]
default-character-set=utf8mb4
```

이후, 아래와 같은 명령어를 통해 db 를 컨테이너로 실행한다.

```
sudo docker build -t baggu-database-deploy .
```

```
sudo docker run --name baggu-database-deploy -d --network cd_network -v my-db-  
volume:/var/lib/mysql -e MARIADB_DATABASE=<사용할 db 이름> -e MARIADB_ROOT_PASSWORD=<비밀번호>  
-p 3306:3306 baggu-database-deploy
```

2-5. Mongodb 설정

Mongodb 의 경우 따로 도커파일을 만들지 않고 도커 커맨드로 실행시킨다.

커맨드는 아래와 같다.

```
sudo docker run -d --name baggu-mongodb-deploy --network cd_network -p 27017:27017 -v  
/your/data/directory:/data/db mongo:5.0.14
```

채팅 서버와 알림 서버에서 Spring Webflux 를 사용하기 때문에 db 에 들어가서 일부분 세팅을 해주어야 한다.

아래 명령어를 통해 mongodb 의 bash 에 접속하자.

```
sudo docker exec -it baggu-mongodb-deploy bash
```

Mongodb 의 bash 에 접속했다면 아래와 같은 명령어를 입력한다.

```
mongo  
  
use chatdb;  
db.createCollection("chat");  
db.runCommand({convertToCapped: 'chat', size: 8192});  
db.createCollection("chatRoom", {capped: false});  
use notifydb;  
db.runCommand({convertToCapped: 'notify', size: 8192});
```

해당 명령어를 입력하면 chat api 와 notify api 에서 데이터를 사용할 수 있다.

3. 빌드 상세 내용

3-1. 백엔드

백엔드의 경우 각종 데이터베이스의 컨테이너나 현재 사용중인 환경에 민감한 정보들이 많이 존재한다. 따라서 이러한 부분들을 환경에 맞게 적절하게 변경해주어야 한다.

빌드 시점에 이를 수정하기 위해서는 docker-compose 파일과 dockerfile 의 RUN 을 이용하여 shell script 를 활용해 조작해야 한다.

우선 application.yml 과 application-SECRET.yml 을 살펴보면 아래와 같다.

application.yml (~/backend/src/main/resources/application.yml)

```
spring:
  profiles:
    include: SECRET
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
    defer-datasource-initialization: false
  datasource:
    driverClassName: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://$database 컨테이너 이름:/baggu?
    serverTimezone=UTC&characterEncoding=utf8&useSSL=false&useUnicode=true
    username: $database 아이디
    password: $database 패스워드
  #console color change
  output:
    ansi:
      enabled: always
  security:
    oauth2:
      client:
        registration:
          kakao:
            authorization-grant-type: authorization_code
            client-id: dcea227af64fcf0366810e14b850e4d6
            redirect-uri: https://$프론트 url 주소/kakaoLogin
            client-authentication-method: POST
            client-name: kakao
        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user_name_attribute: id

#로그설정
logging:
  level:
    org.hibernate.SQL: debug
  # org.hibernate.type: trace
  com:
    amazonaws:
      util:
        EC2MetadataUtils: error

cloud:
  aws:
    region:
      static:
        ap-northeast-2
    stack:
      auto:
        false
  server:
    port: 9999

#swagger
springfox:
  documentation:
    swagger:
      v2:
        path: /swagger/api-docs
```

Database 컨테이너 이름과 사용할 프론트엔드의 URL 은 각 환경마다 서로 다르므로 backend Dockerfile 에서 sed 명령어를 통해 이를 변경한 뒤 컴파일하도록 설정하면 해결할 수 있다.

또한 아래는 application-SECRET.yml 파일의 구조이다.

```
aws_access_key: ${S3 Access key}
aws_private_key: ${S3 private key}
aws_bucket: bagguimg

# 백엔드 로컬 테스트용
# env_staticDir: /src/main/resources/static/

# 서버용
env_staticDir: /app/s3/temp

jwt_secret_key: jwt-temp-secret-key

redis:
  host: 127.0.0.1
  port: 6379
  password: ${Redis 패스워드}
```

위의 파일들에는 직접 발급받은 S3 키와 개인적으로 생성한 Redis 의 패스워드를 입력한다.

해당 부분은 아직 env 로 따로 꺼내지 않았기 때문에 현재 소스 파일에 그대로 반영이 되어있다.

백엔드의 Dockerfile 은 test 에 해당하는 Dockerfile-test 와 deploy 에 해당하는 Dockerfile 두개로 나뉘어져 있다.

만약 application.yml 이 deploy 환경과 일치하도록 설정되어 있다면 아래와 같이 Dockerfile 을 작성할 수 있다.

Dockerfile (~/backend/Dockerfile)

```
# openjdk:11-jdk-slim 을 이미지로 사용한다.
FROM openjdk:11-jdk-slim AS builder

# 작업 디렉토리를 /app으로 설정한다.
WORKDIR /app

# 모든 소스코드 ( 현재 폴더에 있는 소스코드 ) 를 작업 디렉토리에 복사한다.
COPY . .

# gradlew을 사용하여 build한다.
RUN chmod +x gradlew
RUN ./gradlew -x compileTestJava build

# 빌드가 완료된 jar파일을 작업 디렉토리의 root로 가져온다.
FROM openjdk:11-jdk-slim

WORKDIR /app

RUN mkdir /app/s3
RUN mkdir /app/s3/temp

COPY --from=builder /app/build/libs/*.jar app.jar

# 9999 포트를 사용
EXPOSE 9999

# java -jar app.jar을 이용해서 빌드한 파일을 실행한다.
CMD ["java", "-jar", "app.jar"]

~
```

현재 프로젝트에서는 jar 파일 빌드를 Dockerfile 내부에서 할 수 있도록 설정해 두었다.

이미지의 경량화를 위해 MultiStage Build 를 이용한다.

만약 test 환경을 위해 Dockerfile 을 작성해야 한다면 아래와 같이 작성해야 한다.

Dockerfile-test (~/backend/Dockerfile-test)

```
# openjdk:11-jdk-slim 을 이미지로 사용한다.
FROM openjdk:11-jdk-slim AS builder

# 작업 디렉토리를 /app으로 설정한다.
WORKDIR /app

# 모든 소스코드 ( 현재 폴더에 있는 소스코드 ) 를 작업 디렉토리에 복사한다.
COPY . .

# gradlew을 사용하여 build한다.
RUN sed -i 's/mariadb:\\\\{배포용 db 컨테이너 이름}/mariadb:\\\\{테스트용 db 컨테이너 이름}/g' /app/src/main/resources/application.yml
RUN sed -i 's/host: {배포용 redis 컨테이너 이름}/host: {테스트용 redis 컨테이너 이름}/g' /app/src/main/resources/application-SECRET.yml
RUN chmod +x gradlew
RUN ./gradlew -x compileTestJava build

# 빌드가 완료된 jar파일을 작업 디렉토리의 root로 가져온다.
FROM openjdk:11-jdk-slim

WORKDIR /app

RUN mkdir /app/s3
RUN mkdir /app/s3/temp

COPY --from=builder /app/build/libs/*.jar app.jar

# 9999 포트를 사용
EXPOSE 9999

# java -jar app.jar을 이용해서 빌드한 파일을 실행한다.
CMD ["java", "-jar", "app.jar"]
```

위와 같이 sed 명령어를 통해 환경에 맞는 컨테이너와 적절하게 통신할 수 있도록 설정해야 한다.

3-2. 프론트엔드

Dockerfile (~/frontend/Dockerfile)

```
# node 18 버전을 사용한다.
FROM node:18 as builder

# 작업 디렉토리를 /usr/src/app으로 설정.
RUN mkdir /usr/src/app
WORKDIR /usr/src/app

# 환경 변수 PATH를 설정한다.
ENV PATH /usr/src/app/node_modules/.bin:$PATH

# package.json, package-lock.json을 복사한다.
COPY package*.json ./

# 의존성 인스톨.
RUN npm install --silent

# 나머지 파일을 복사한다.
COPY . .

# npm Build
RUN npm run build

# nginx를 서버로 설정
FROM nginx:stable-alpine

# 컨테이너의 작업 디렉토리를 /app으로 변경
WORKDIR /app

# 컨테이너에 build 폴더 복사
RUN mkdir build
COPY --from=builder /usr/src/app/build ./build

# nginx의 설정을 개인 설정으로 변경
RUN rm -rf /etc/nginx/conf.d/default.conf
COPY ./nginx.conf /etc/nginx/conf.d

# 포트 번호 명시
EXPOSE 3000

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

3-3. Elasticsearch

Dockerfile (~/elasticsearch/Dockerfile)

```
# 작성
ARG ELK_VERSION

FROM docker.elastic.co/elasticsearch/elasticsearch:${ELK_VERSION}

RUN elasticsearch-plugin install analysis-nori
```

3-4. notify Server

Dockerfile (~/notify/Dockerfile)

```
# openjdk:11-jdk-slim 을 이미지로 사용한다.
FROM openjdk:11-jdk-slim AS builder

# 작업 디렉토리를 /app으로 설정한다.
WORKDIR /app

# 모든 소스코드 ( 현재 폴더에 있는 소스코드 ) 를 작업 디렉토리에 복사한다.
COPY . .

# gradlew을 사용하여 build한다.
RUN chmod +x gradlew
RUN ./gradlew clean build

# 빌드가 완료된 jar파일을 작업 디렉토리의 root로 가져온다.
FROM openjdk:11-jdk-slim

COPY --from=builder /app/build/libs/*.jar app.jar

# 9999 포트를 사용
EXPOSE 9998

# java -jar app.jar을 이용해서 빌드한 파일을 실행한다.
CMD ["java", "-jar", "app.jar"]
```

3-5. chat Server

Dockerfile (~/chat/Dockerfile)

```
# openjdk:11-jdk-slim 을 이미지로 사용한다.
FROM openjdk:11-jdk-slim AS builder

# 작업 디렉토리를 /app으로 설정한다.
WORKDIR /app

# 모든 소스코드 ( 현재 폴더에 있는 소스코드 ) 를 작업 디렉토리에 복사한다.
COPY . .

# gradlew을 사용하여 build한다.
RUN chmod +x gradlew
RUN ./gradlew clean build

# 빌드가 완료된 jar파일을 작업 디렉토리의 root로 가져온다.
FROM openjdk:11-jdk-slim

COPY --from=builder /app/build/libs/*.jar app.jar

# 9999 포트를 사용
EXPOSE 9997

# java -jar app.jar을 이용해서 빌드한 파일을 실행한다.
CMD ["java", "-jar", "app.jar"]
```

3-6. redis

Dockerfile (~/redis/Dockerfile-deploy)

```
# redis 7.0.7 사용
FROM redis:7.0.7

# 시작 지점설정
RUN mkdir -p /etc/redis

# 레디스 설정 복사
COPY ./config/redis-deploy.conf /etc/redis/redis.conf

# 설정으로 시작하게 만들음.
CMD ["redis-server", "/etc/redis/redis.conf"]
```

redis.conf(~/redis/config/redis.conf)

```
save ""

bind baggu-redis-deploy 127.0.0.1
requirepass
"8B4FEDDBD6709F57F46AADAAB0E18CDA964413A4208D828C7D68B3E7028DE0E6758C9C85A71AD65F8B5D3672
7CB1DEBF33E854486BEBEACEE0344200887C52CB"
```

3-7. docker-compose.yml

docker-compose.yml (~/docker-compose.yml)

```
version: '3.8'
services:
  baggu-backend:
    container_name: baggu-backend
    build:
      dockerfile: Dockerfile
      context: ./backend
    environment:
      - SPRING_DATASOURCE_USERNAME=root
      - SPRING_DATASOURCE_PASSWORD=A708
      - SPRING_DATASOURCE_URL=jdbc:mariadb://baggu-database-
deploy:3306/baggu
    ports:
      - "9999:9999"
    restart: always
    depends_on:
      - baggu-redis-deploy
      - baggu-elasticsearch-deploy
  baggu-redis-deploy:
    container_name: baggu-redis-deploy
    build:
      dockerfile: Dockerfile-deploy
      context: ./redis
    ports:
      - "6382:6379"
    restart: always
  baggu-chat:
    container_name: baggu-chat
    build:
      dockerfile: Dockerfile
      context: ./chat
    ports:
      - "9997:9997"
    restart: always
  baggu-notify:
    container_name: baggu-notify
    build:
      dockerfile: Dockerfile
      context: ./notify
    ports:
      - "9998:9998"
    restart: always
  baggu-frontend:
    container_name: baggu-frontend
    build:
      dockerfile: Dockerfile
      context: ./frontend
    ports:
      - "3000:3000"
    restart: always
  baggu-elasticsearch-deploy:
    container_name: baggu-elasticsearch-deploy
    deploy:
      resources:
        limits:
          memory: 3G
    environment:
      - node.name=single-node
      - cluster.name=baggudeploy
      - discovery.type=single-node
    ports:
      - "9202:9200"

    build:
      dockerfile: Dockerfile
      context: ./elasticsearch
      args:
        ELK_VERSION: 7.15.2
    restart: always
networks:
  default:
    external:
      name: cd_network
```

빌드 하기 위해서는 프로젝트의 루트 폴더에서 아래와 같은 명령어를 입력한다.

```
sudo docker-compose -f docker-compose.yml up -d --build
```

이미 빌드가 되어 있는 컴포즈 컨테이너를 일괄적으로 종료하고자 하면 아래의 명령어를 입력한다.

```
sudo docker-compose -f docker-compose.yml down --rmi all
```

만약 테스트 환경에서 빌드 스크립트를 다르게 적용하고 싶다면, docker-compose 과정에서 -f 를 통해 파일을 설정하면 된다.

```
ci-docker-compose.yml(~/ci-docker-compose.yml)
```



```
version: '3.8'
services:
  baggu-database-test:
    container_name: baggu-database-test
    build:
      dockerfile: Dockerfile
      context: ./database
    environment:
      - MARIADB_DATABASE=baggu
      - MARIADB_ROOT_PASSWORD=A708
    volumes:
      - ./database/config:/etc/mysql/conf.d
    ports:
      - "3307:3306"
    restart: always
  baggu-redis-test:
    container_name: baggu-redis-test
    build:
      dockerfile: Dockerfile
      context: ./redis
    ports:
      - "6380:6379"
    restart: always
  baggu-backend-test:
    container_name: baggu-backend-test
    build:
      dockerfile: Dockerfile-test
      context: ./backend
    environment:
      - SPRING_DATASOURCE_USERNAME=root
      - SPRING_DATASOURCE_PASSWORD=A708
      - SPRING_DATASOURCE_URL=jdbc:mariadb://baggu-database-
test:3306/baggu
    ports:
      - "8888:9999"
    depends_on:
      - baggu-database-test
    restart: always
  baggu-frontend-test:
    container_name: baggu-frontend-test
    build:
      dockerfile: Dockerfile
      context: ./frontend
    ports:
      - "3001:3000"
    restart: always
```

4. 외부 서비스

4-1. AWS S3 / CloudFront

1) 계정 설정 및 백엔드 프로젝트 연동

키에 대한 외부 접근의 위험을 막기 위해 S3와 CloudFront의 접근 권한만을 가진 IAM 사용자를 생성하여 등록한다.

IAM > 사용자 > s3admin

s3admin

삭제

요약

ARN arn:aws:iam::249740145457:user/s3admin	콘솔 액세스 비활성화됨	액세스 키 1 AKIATUJNMGMY47ABRKUF - Active 🟢 오늘 사용됨. 8일 기준.
생성됨 February 07, 2023, 09:32 (UTC+09:00)	마지막 콘솔 로그인 -	액세스 키 2 활성화되지 않음

권한 | 그룹 | 태그 | 보안 자격 증명 | 액세스 관리자

권한 정책 (2)

사용자에게 직접 연결된 정책을 통해 또는 그룹을 통해 권한을 정의합니다.



제거

권한 추가 ▼

🔍 정책 찾기

< 1 > ⚙️

<input type="checkbox"/>	정책 이름	▲ 유형	▼ 연결 방식:
<input type="checkbox"/>	AmazonS3FullAccess	AWS 관리형	직접
<input type="checkbox"/>	CloudFrontFullAccess	AWS 관리형	직접

S3 서버에 연결하기 위한 라이브러리 의존성을 주입한다.

```
implementation 'org.springframework.cloud:spring-cloud-starter-aws:2.0.1.RELEASE'
```

AWS Configuration을 생성하기 위한 access key와 private key는 application-SECRET.yml로 별도 관리하며, profiles 속성을 통해 주입시킨다.

```
1 aws_access_key: AKIATUJNMGMY47ABRKUF
2 aws_private_key:
3 aws_bucket: bagguimg
```

```
1 spring:
2   profiles:
3     include: SECRET
```

2) S3 버킷 생성 및 설정

서버에서 버킷에 접근하여 이미지를 직접해야하므로, 퍼블릭 액세스를 허용한다.

퍼블릭 액세스 차단(버킷 설정)

퍼블릭 액세스는 ACL(엑세스 제어 목록), 버킷 정책, 액세스 지점 정책 또는 모두를 통해 버킷 및 객체에 부여됩니다. 모든 S3 버킷 및 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 [모든 퍼블릭 액세스 차단]을 활성화합니다. 이 설정은 이 버킷 및 해당 액세스 지점에만 적용됩니다. AWS에서는 [모든 퍼블릭 액세스 차단]을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 버킷 또는 내부 객체에 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. [자세히 알아보기](#)

편집

모든 퍼블릭 액세스 차단

⚠ 비활성

▶ 이 버킷의 개별 퍼블릭 액세스 차단 설정

다음은 생성한 버킷 정책이다. 앞서 생성한 S3 IAM 에 대해선 모든 액션을 허용하고, 클라우드 프론트 ARN 에 대해선 파일 읽기 액션을 허용한다.

```
1  {
2      "Version": "2012-10-17",
3      "Id": "Policy1675730718503",
4      "Statement": [
5          {
6              "Sid": "Stmt1675730717312",
7              "Effect": "Allow",
8              "Principal": {
9                  "AWS": "arn:aws:iam::249740145457:user/s3admin"
10             },
11             "Action": "s3:*",
12             "Resource": "arn:aws:s3:::bagguimg/*"
13         },
14         {
15             "Sid": "Stmt1675730717313",
16             "Effect": "Allow",
17             "Principal": {
18                 "Service": "cloudfront.amazonaws.com"
19             },
20             "Action": "s3:GetObject",
21             "Resource": "arn:aws:s3:::bagguimg/*",
22             "Condition": {
23                 "StringEquals": {
24                     "AWS:SourceArn": "arn:aws:cloudfront::249740145457:distribution/E3LS0Y4W2645D0"
25                 }
26             }
27         }
28     ]
29 }
```

3) CloudFront 생성

다음은 CloudFront 상세 설정이다. 접근은 아시아까지, 메소드는 HTTP GET 에 대해 허용한다. AWS 의 Route53 를 사용하여 연결하지 않았기 때문에, CloudFront 에 대한 대체 도메인은 설정하지 않았다. 원본 이미지는 이전에 생성한 S3 버킷을 생성하도록 한다.

설정			편집
설명 -	대체 도메인 이름 -	표준 로깅 끄기	
가격 분류 북미, 유럽, 아시아, 중동 및 아프리카에서 사용		쿠키 로깅 끄기	
지원되는 HTTP 버전 HTTP/2, HTTP/1.1, HTTP/1.0		기본 루트 객체 -	
AWS WAF -			

4-2. 카카오 로그인 API

1) Kakao developers 에 애플리케이션 등록 후 로그인 API 설정

Kakao developers(<https://developers.kakao.com/console/app>)에 애플리케이션을 등록한다.

전체 애플리케이션 (1)		애플리케이션 이름
<div><div>+</div>애플리케이션 추가하기</div>		
<div><div>B</div><div>바꾸바꾸</div><div>ID 844852 OWNER Web</div></div>		

카카오 로그인 기능을 활성화하고, 동의항목에서 닉네임과 카카오계정(이메일)을 설정한다.

항목 이름	ID	상태	
닉네임	profile_nickname	● 필수 동의	설정
프로필 사진	profile_image	● 사용 안함	설정
카카오계정(이메일)	account_email	● 선택 동의	설정

다음은 설정된 RedirectURI 이다. 순서대로 백엔드 로컬 테스트용, 프론트엔드 로컬 테스트용, 배포 서버용, 테스트 서버용이다.

Redirect URI

삭제

수정

Redirect URI	http://localhost:9999/baggu/auth/callback/kakao http://localhost:3000/kakaoLogin https://baggu.shop/kakaoLogin https://test.baggu.shop/kakaoLogin
--------------	--

- 카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다. (최대 10개)
- REST API로 개발하는 경우 필수로 설정해야 합니다.

2) Spring Security OAuth2 사용을 위한 Configuration 등록

Spring Security OAuth2 를 사용하기 위해선 OAuth2 기능 제공 서버의 정보를 application.yml 에 작성해야한다. 다음과 같이 작성한다. 카카오의 경우엔 Spring Security 자체에서 provider 정보를 제공하지 않기 때문에, 따로 관련 uri 를 함께 작성한다.

```

31 security:
32   oauth2:
33     client:
34       registration:
35         kakao:
36           authorization-grant-type: authorization_code
37           client-id: dcea227af64fcf0366810e14b850e4d6
38           #백엔드 로컬 테스트
39           # redirect-uri: http://localhost:9999/baggu/auth/callback/kakao
40           #프론트엔드 로컬 테스트
41           # redirect-uri: http://localhost:3000/kakaoLogin
42           #테스트 서버용
43           redirect-uri: https://test.baggu.shop/kakaoLogin
44           #배포 서버용
45           # redirect-uri: http://baggu.shop/kakaoLogin
46           client-authentication-method: POST
47           client-name: kakao
48       provider:
49         kakao:
50           authorization_uri: https://kauth.kakao.com/oauth/authorize
51           token_uri: https://kauth.kakao.com/oauth/token
52           user-info-uri: https://kapi.kakao.com/v2/user/me
53           user_name_attribute: id

```