

**바꾸바꾸**

23.01.03 ~ 23.02.17

**포팅 매뉴얼**

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
5. **프로젝트 기술 스택**
6. 이슈관리: Jira
7. 형상관리: Git
8. 커뮤니케이션: Mattermost, Webex, notion
9. 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
10. 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 \  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  $(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

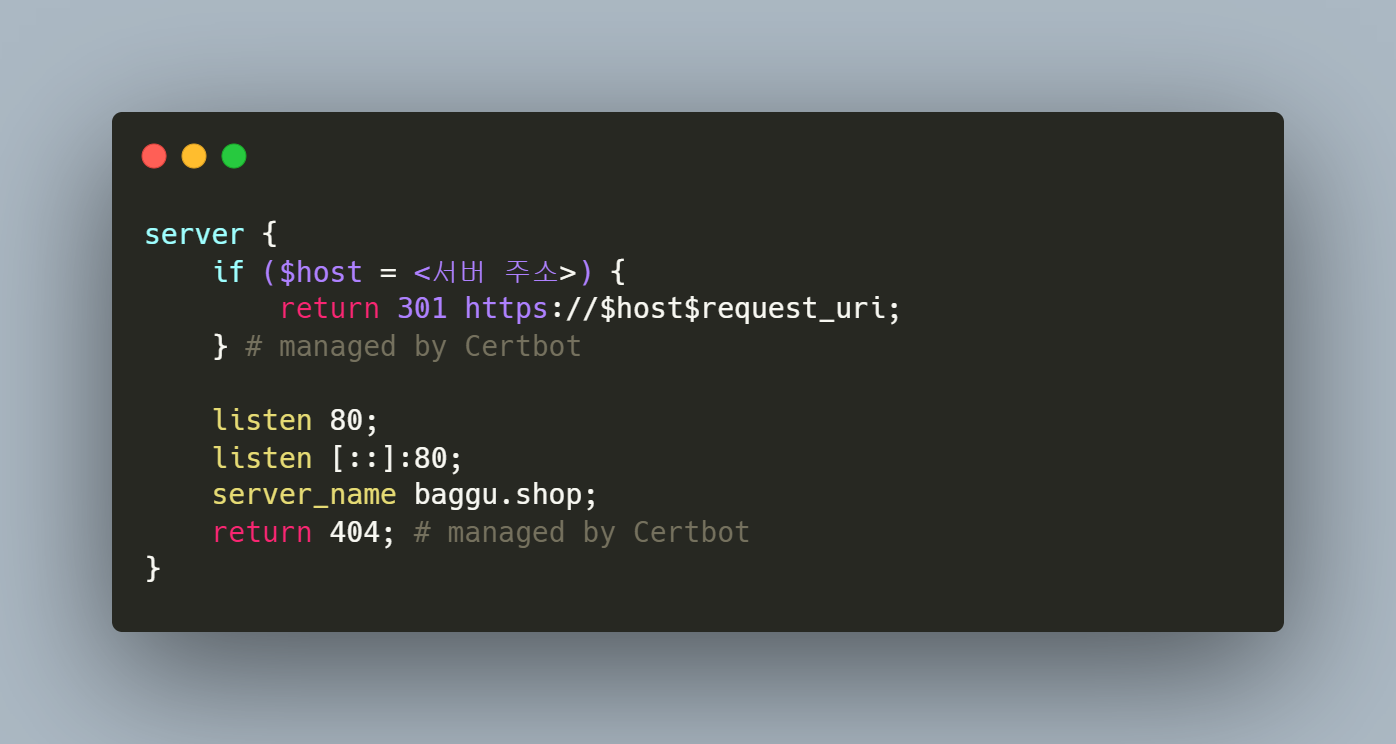
|  |
| --- |
| sudo apt install nginx |

4) nginx 설정

|  |
| --- |
| cd /etc/nginx/conf.d  sudo vim default.conf |

jenkins의 nginx 설정은 아래의 링크를 참조하여 진행한다.

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



5) Certbot nginx에 적용

|  |
| --- |
| sudo certbot –nginx |

이 과정을 거치게 되면 nginx의 default.conf에 자동으로 ssl인증서가 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 jenkins를 사용하려면 아래의 과정을 따른다.

젠킨스 이미지 다운로드

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

jenkins를 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 jenkins명령어를 통해 jenkins의 암호를 확인한다.

Nginx로 라우팅처리한 url을 통해 jenkins에 접속한다.

Dashboard > Jenkins 관리 > Plugin Manager로 접속한 뒤 Available plugins에서 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 EC2에서 아래의 명령어를 통해 Jenkins 컨테이너의 bash로 접근한다.

|  |
| --- |
| sudo docker exec –it jenkins bash |

jenkins 내부에서 docker에 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 |

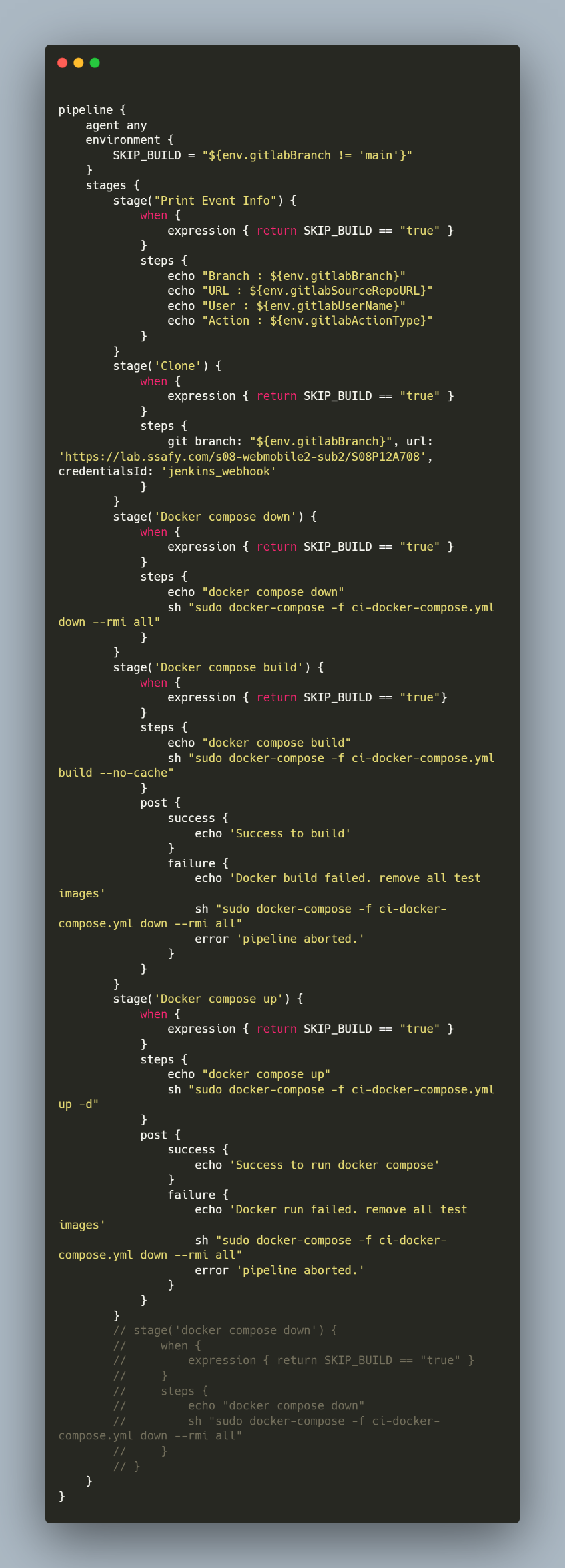
docker-compose를 사용하기 위해 아래 명령어를 추가로 입력한다.

|  |
| --- |
| 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.



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

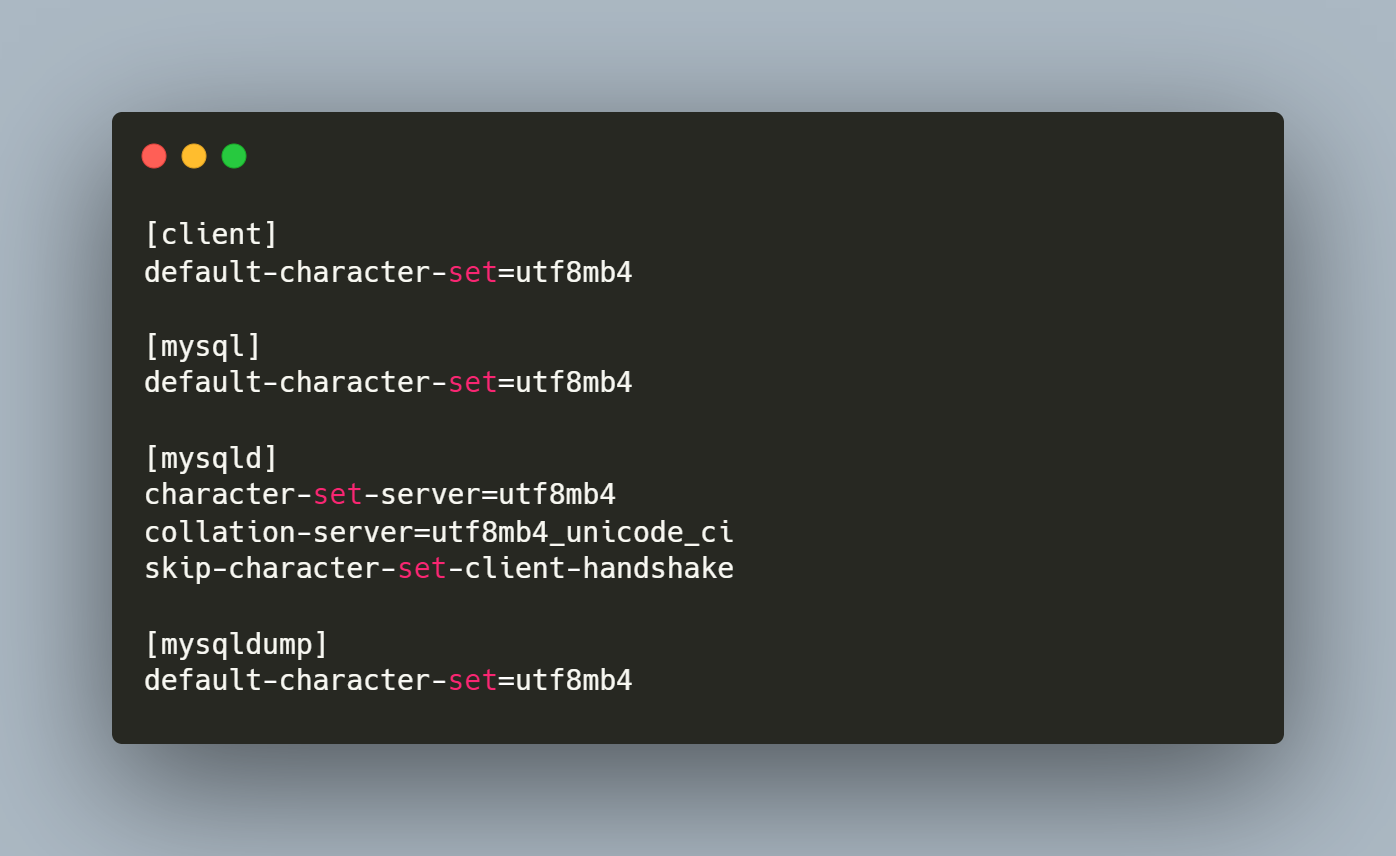
**2-4. Mariadb 설정**

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

Dockerfile (~/database/Dockerfile)



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



이후, 아래와 같은 명령어를 통해 db를 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 db이름> -e MARIADB\_ROOT\_PASSWORD=<비밀번호> -p 3306:3306 baggu-database-deploy |

**2-5. Mongodb 설정**

Mongodb의 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 Webflux를 사용하기 때문에 db에 들어가서 일부분 세팅을 해주어야 한다.

아래 명령어를 통해 mongodb의 bash에 접속하자.

|  |
| --- |
| sudo docker exec –it baggu-mongodb-deploy bash |

Mongodb의 bash에 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 api와 notify api에서 데이터를 사용할 수 있다.

**3. 빌드 상세 내용**

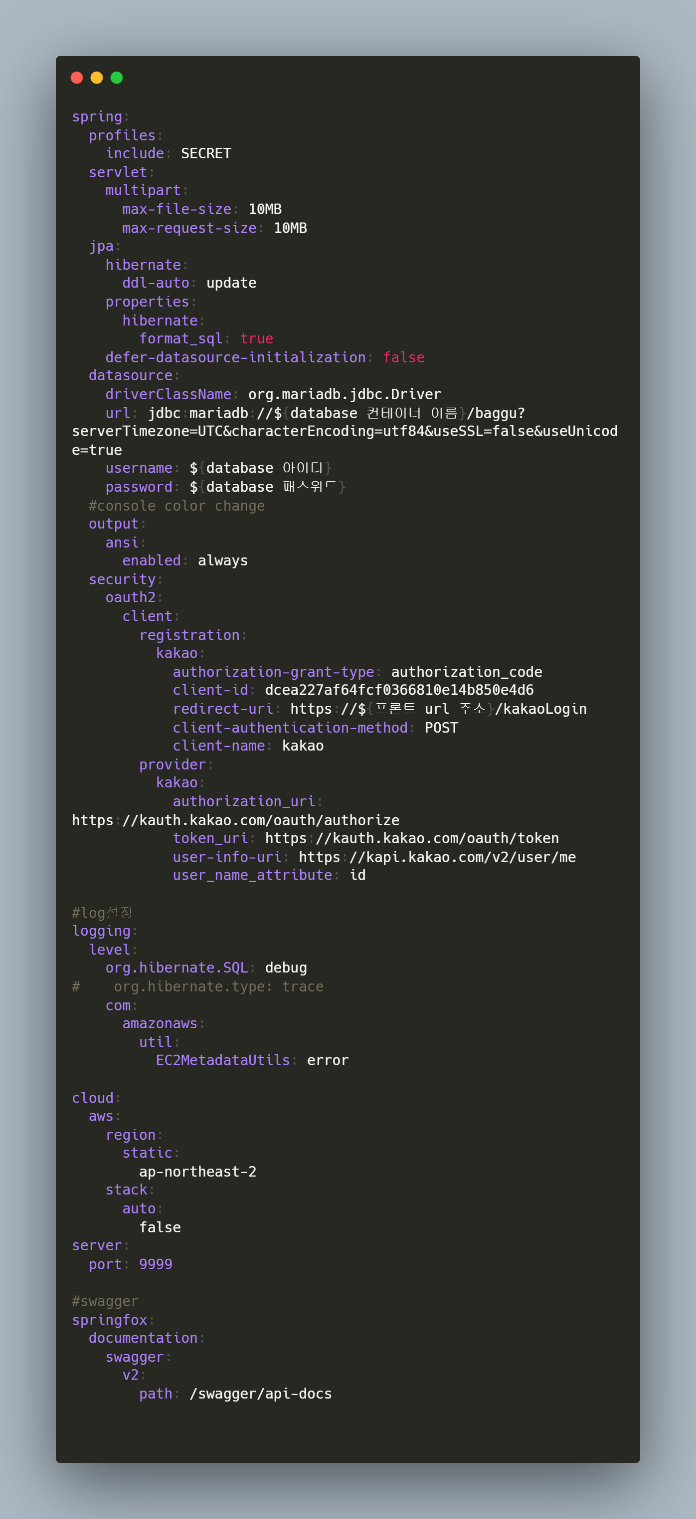
**3-1. 백엔드**

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

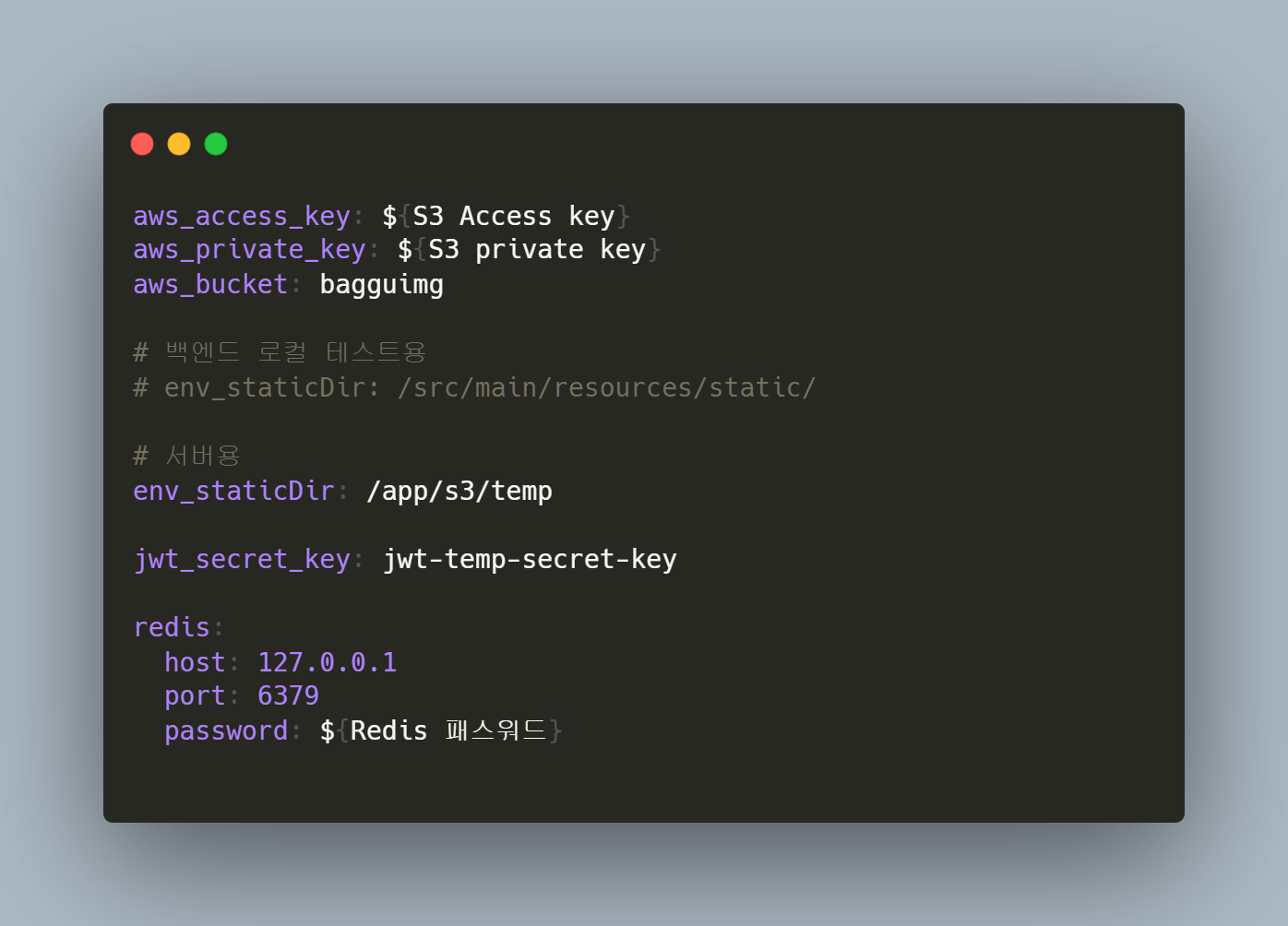
빌드 시점에 이를 수정하기 위해서는 docker-compose 파일과 dockerfile의 RUN을 이용하여 shell script를 활용해 조작해야 한다.

우선 application.yml과 application-SECRET.yml을 살펴보면 아래와 같다.

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



Database 컨테이너 이름과 사용할 프론트엔드의 URL은 각 환경마다 서로 다르므로 backend Dockerfile에서 sed 명령어를 통해 이를 변경한 뒤 컴파일하도록 설정하면 해결할 수 있다.

또한 아래는 application-SECRET.yml 파일의 구조이다.

위의 파일들에는 직접 발급받은 S3 키와 개인적으로 생성한 Redis의 패스워드를 입력한다.

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

백엔드의 Dockerfile은 test에 해당하는 Dockerfile-test와 deploy에 해당하는 Dockerfile 두개로 나뉘어져 있다.

만약 application.yml이 deploy 환경과 일치하도록 설정되어 있다면 아래와 같이 Dockerfile을 작성할 수 있다.

Dockerfile ( ~/backend/Dockerfile )



현재 프로젝트에서는 jar파일 빌드를 Dockerfile 내부에서 할 수 있도록 설정해 두었다.

이미지의 경량화를 위해 MultiStage Build를 이용한다.

만약 test환경을 위해 Dockerfile을 작성해야 한다면 아래와 같이 작성해야 한다.

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



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

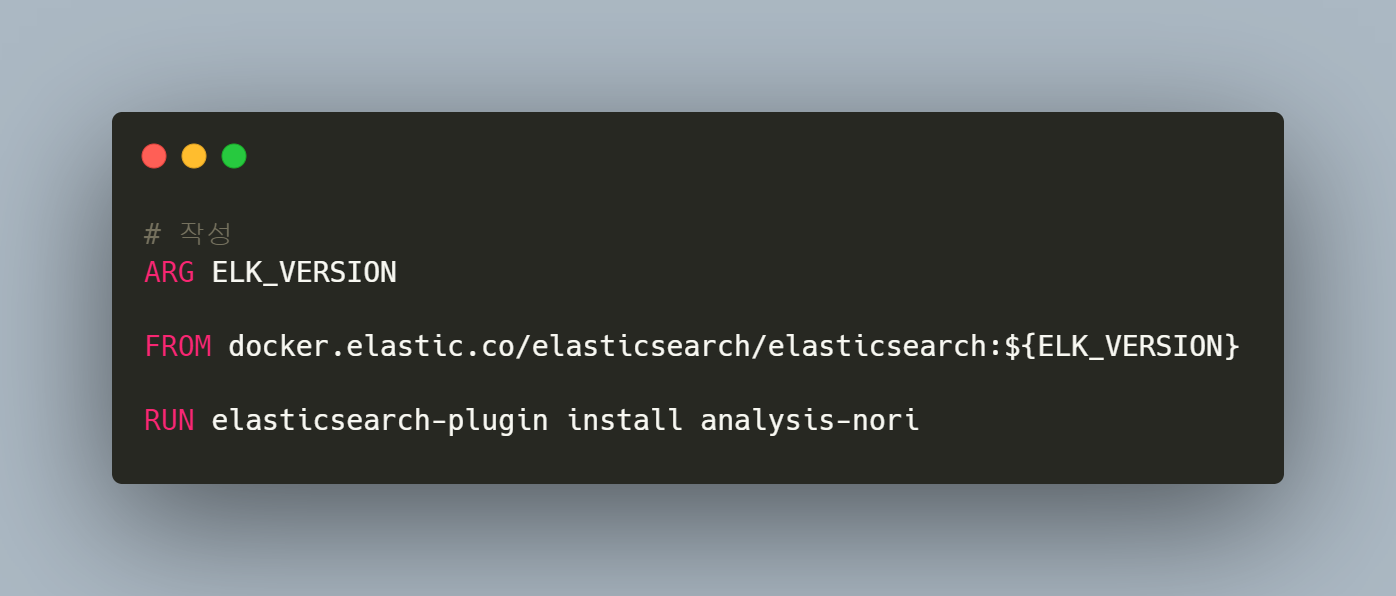
**3-2. 프론트엔드**

Dockerfile ( ~/frontend/Dockerfile )



**3-3. ElasticSearch**

Dockerfile ( ~/elasticsearch/Dockerfile )

****

**3-4. notify Server**

Dockerfile ( ~/notify/Dockerfile )

****

**3-5. chat Server**

Dockerfile ( ~/chat/Dockerfile )

**3-6. redis**

Dockerfile ( ~/redis/Dockerfile-deploy )

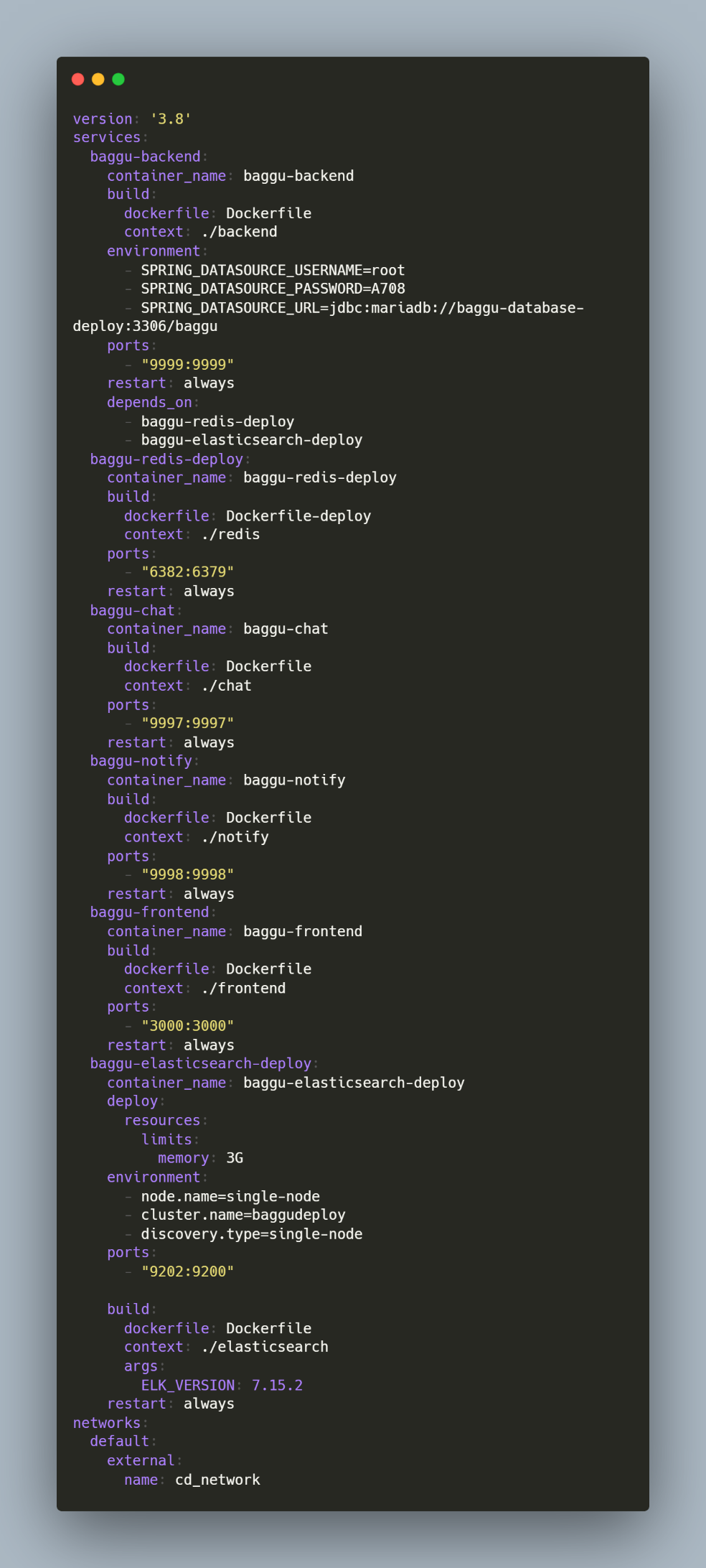


redis.conf(~/redis/config/redis.conf)

|  |
| --- |
| save ""  bind baggu-redis-deploy 127.0.0.1  requirepass "8B4FEDDBD6709F57F46AADAAB0E18CDA964413A4208D828C7D68B3E7028DE0E6758C9C85A71AD65F8B5D36727CB1DEBF33E854486BEBEACEE0344200887C52CB" |

**3-7. docker-compose.yml**

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



빌드 하기 위해서는 프로젝트의 루트 폴더에서 아래와 같은 명령어를 입력한다.

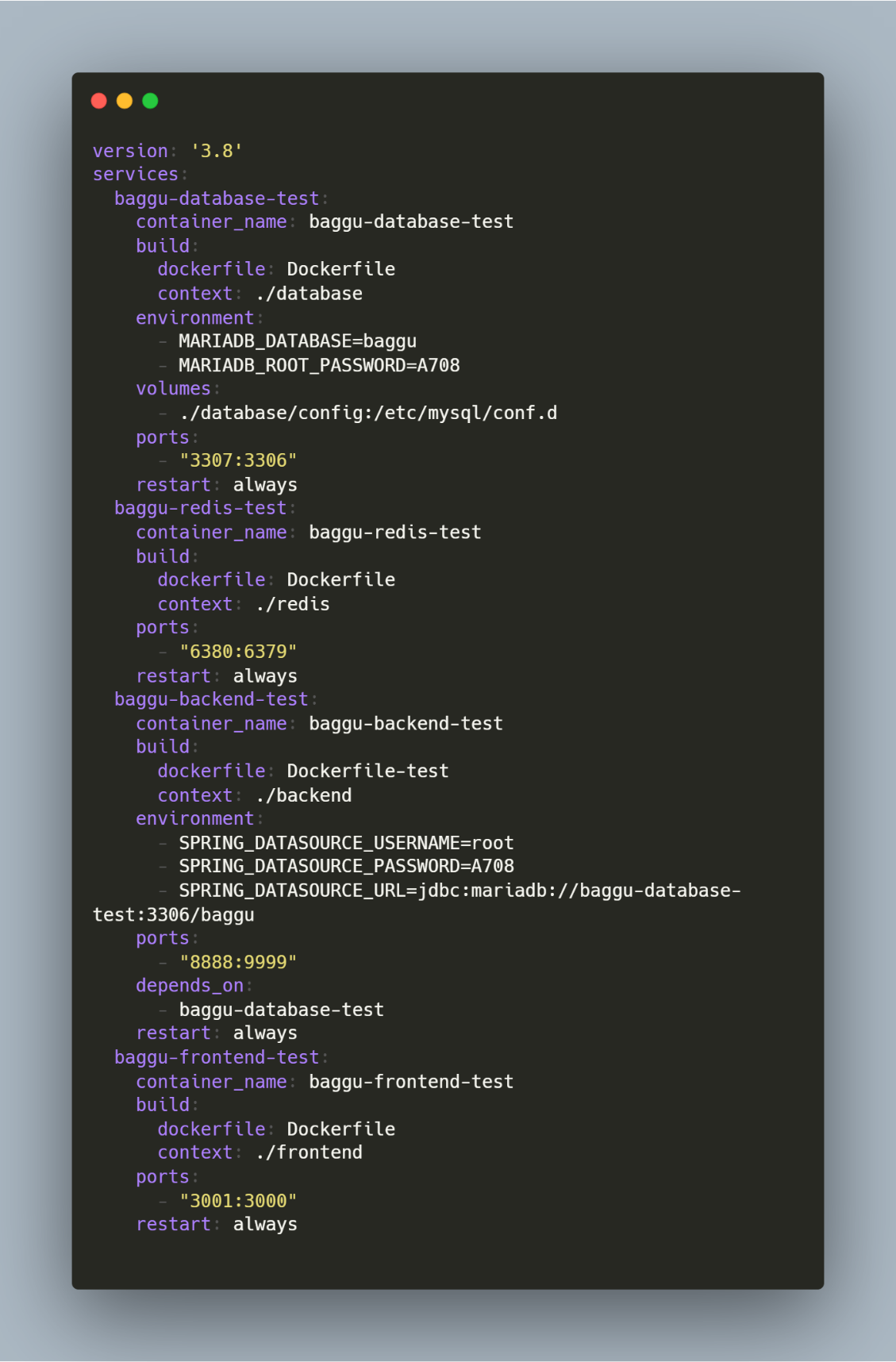
|  |
| --- |
| sudo docker-compose –f docker-compose.yml up –d –-build |

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 –f를 통해 파일을 설정하면 된다.

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

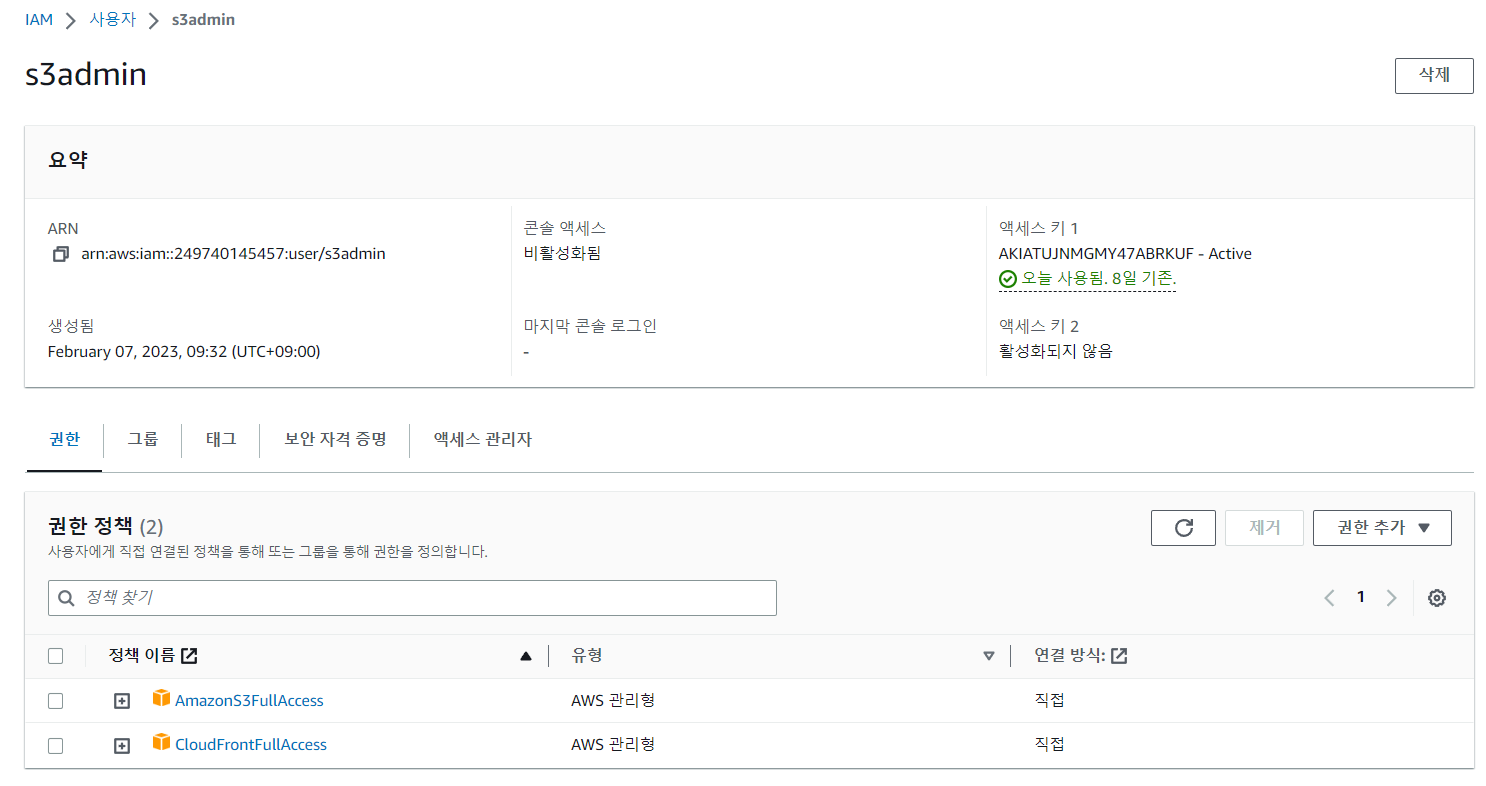


**4. 외부 서비스**

**4-1. AWS S3 / CloudFront**

1. 계정 설정 및 백엔드 프로젝트 연동

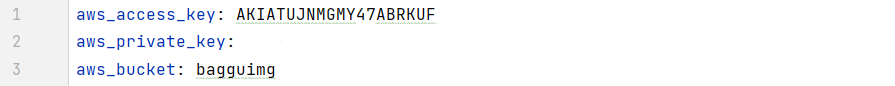
키에 대한 외부 접근의 위험을 막기 위해 S3와 CloudFront의 접근 권한만을 가진 IAM 사용자를 생성하여 등록한다.



S3 서버에 연결하기 위한 라이브러리 의존성을 주입한다.

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

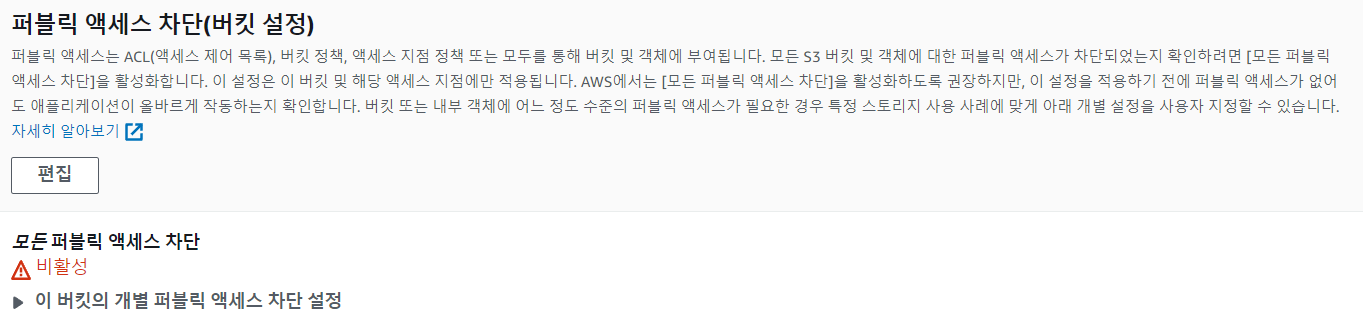
AWS Configuration을 생성하기 위한 access key와 private key는 application-SECRET.yml으로 별도 관리하며, profiles 속성을 통해 주입시킨다.





1. S3 버킷 생성 및 설정

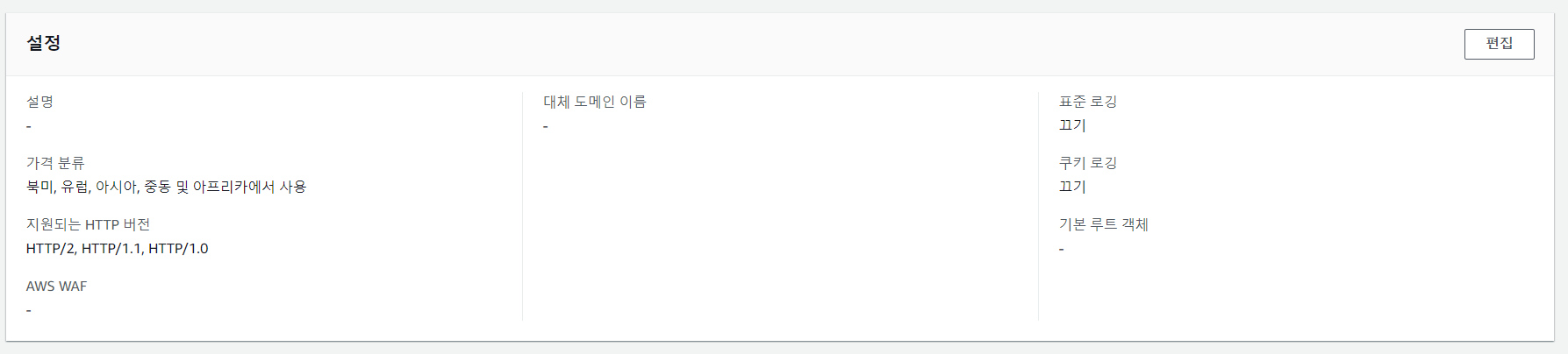
서버에서 버킷에 접근하여 이미지를 직접해야하므로, 퍼블릭 액세스를 허용한다.



다음은 생성한 버킷 정책이다. 앞서 생성한 S3 IAM에 대해선 모든 액션을 허용하고, 클라우드 프론트 ARN에 대해선 파일 읽기 액션을 허용한다.

1. CloudFront 생성

다음은 CloudFront 상세 설정이다. 접근은 아시아까지, 메소드는 HTTP GET에 대해 허용한다. AWS의 Route53를 사용하여 연결하지 않았기 때문에, CloudFront에 대한 대체 도메인은 설정하지 않았다. 원본 이미지는 이전에 생성한 S3 버킷을 생성하도록 한다.



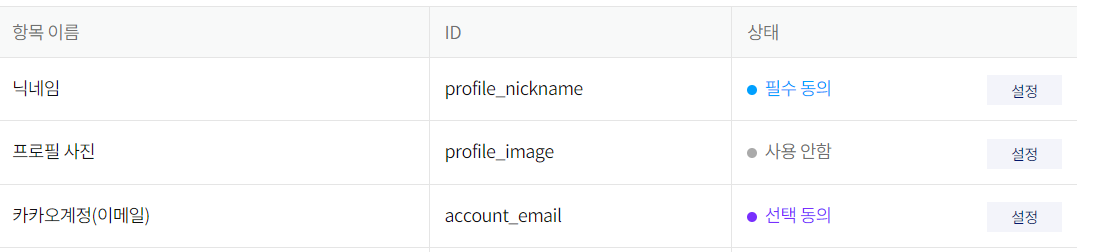
**4-2. 카카오 로그인 API**

1. Kakao developers에 애플리케이션 등록 후 로그인 API 설정

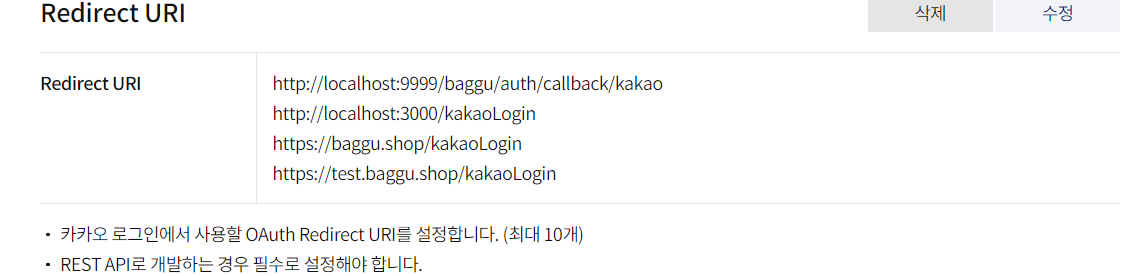
Kakao developers(https://developers.kakao.com/console/app)에 애플리케이션을 등록한다.



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



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



1. Spring Security OAuth2 사용을 위한 Configuration 등록

Spring Security OAuth2를 사용하기 위해선 OAuth2 기능 제공 서버의 정보를 application.yml에 작성해야한다. 다음과 같이 작성한다. 카카오의 경우엔 Spring Security 자체에서 provider 정보를 제공하지 않기 때문에, 따로 관련 uri를 함께 작성한다.

