



# 포팅 메뉴얼

## 1. 운영환경

### 1-1. 사용한 버전

## 2. Infra

### 2-1. 사용한 포트 정보

### 2-2. DATABASE

### 2-3. Storage

### 2-4. WebServer

### 2-5. Jenkins

## 3. Backend

### 3-1. Spring 설정파일

### 3-2 Dockerfile

### 3-3 Jenkinsfile

## 4. Frontend

### 3-1 Dockerfile

### 3-3 Jenkinsfile

## 1. 운영환경

### 1-1. 사용한 버전

분류	환경	버전
BackEnd	Spring Boot	2.7.9
	JPA	2.7.9
	JDK	openjdk - 11.0.17
	Spring Security	5.7.6
	OAuth2	2.7.9
	QueryDSL	5.0.0
FrontEnd	React.js	18.2.0
	Next.js	13.2.4
	Redux-toolkit	1.9.3
	node.js	18.12.0
	styled-components	5.3.9
	tailwindCSS	3.2.7
	typescript	4.9.5
Data	python	3.9
	FastAPI	0.92.0
DataBase	MySQL	8.0.30
	Redis	7.0.8
Infra	Docker	23.0.0
	jenkin	2.375.2
	ubuntu	20.04 LTS

## 2. Infra

## 2-1. 사용한 포트 정보

ufw를 사용해서 포트를 개방해준다.

Port	서비스 명	개방여부
22/TCP	SSH	O
80	NGINX	O
443	NGINX	O
9090	Jenkins	O
8080	spring boot	X
3000	React&nextJS	X

```
$sudo ufw allow <포트번호> # 포트 개방  
$sudo ufw enable # 방화벽 구동
```

## 2-2. DATABASE

### ▼ 과정

※ MySQL은 AWS RDS로 진행함.

#### 1. AWS RDS에서 MySQL을 선택.

### 데이터베이스 생성


#### 데이터베이스 생성 방식 선택


☒ 표준 생성  
가용성, 보안, 백업 및 유지 관리에 대한 옵션을 포함하여 모든 구성 옵션을 설정합니다.


☐ 손쉬운 생성  
권장 모범 사례 구성을 사용합니다. 일부 구성 옵션은 데이터베이스를 생성한 후 변경할 수 있습니다.


#### 엔진 옵션


##### 엔진 유형 정보


☐ Amazon Aurora  


☒ MySQL  


☐ MariaDB  


☐ PostgreSQL  


☐ Oracle  


☐ Microsoft SQL Server  


##### 에디션

☒ MySQL Community

## 2. mysql - 8.0.30 선택

에디션

☒ MySQL Community

 **알려진 문제/제한 사항**  
알려진 문제/제한 사항 [\[?\]](#)을 검토하여 특정 데이터베이스 버전과 발생할 수 있는 호환성 문제를 확인하세요.

▼ 필터 숨기기

☐ **다중 AZ DB 클러스터를 지원하는 버전 표시** [정보](#)  
기본 DB 인스턴스 1개와 읽기 가능한 대기 DB 인스턴스 2개로 다중 AZ DB 클러스터를 생성합니다. 다중 AZ DB 클러스터는 최대 2배 빠른 트랜잭션 커밋 지연 시간과 일반적으로 35초 미만의 자동 장애 조치를 제공합니다.

☐ **Amazon RDS Optimized Writes를 지원하는 버전 표시** [정보](#)  
Amazon RDS Optimized Writes는 추가 비용 없이 쓰기 처리량(throughput)을 최대 2배 늘립니다.

엔진 버전

MySQL 8.0.30 ▼

## 3. 프리티어를 선택해서 구성

**템플릿**  
해당 사용 사례를 충족하는 샘플 템플릿을 선택하세요.

☐ **프로덕션**  
고가용성 및 빠르고 일관된 성능을 위해 기본값을 사용하세요.

☐ **개발/테스트**  
이 인스턴스는 프로덕션 환경 외부에서 개발 용도로 마련되었습니다.

☒ **프리 티어**  
RDS 프리 티어를 사용하여 새로운 애플리케이션을 개발하거나, 기존 애플리케이션을 테스트하거나 Amazon RDS에서 실무 경험을 쌓을 수 있습니다. [정보](#)

## 4. DB 인스턴스 이름과, 사용자 이름, 암호 설정

## 설정

### DB 인스턴스 식별자 정보

DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.

`db-instance-identifier`

DB 인스턴스 식별자는 대소문자를 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자여야 합니다. 하이픈 2개가 연속될 수 없습니다. 하이픈으로 끝날 수 없습니다.

### ▼ 자격 증명 설정

#### 마스터 사용자 이름 정보

DB 인스턴스의 마스터 사용자에 로그인 ID를 입력하세요.

1~16자의 영숫자. 첫 번째 문자는 글자여야 합니다.

☐ AWS Secrets Manager에서 마스터 보안 인증 관리 - 신규

Secrets Manager에서 마스터 사용자 보안 인증을 관리합니다. RDS는 사용자 대신 암호를 생성하고 수명 주기 동안 이를 관리할 수 있습니다.

☐ 암호 자동 생성

Amazon RDS에서 사용자를 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.

#### 마스터 암호 정보

제약 조건: 8자 이상의 인쇄 가능한 ASCII 문자. 다음은 포함할 수 없습니다. /(슬래시), '(작은따옴표)', "(큰따옴표) 및 @(앳 기호).

#### 마스터 암호 확인 정보

⇒ 위에서 언급한 내용만 설정 후, 나머지는 AWS에서 제공하는 기본 설정을 따른다.

## 2-3. Storage

### ▼ 과정

※ 파일 스토리지는 AWS S3를 사용한다.

#### 1. 버킷 생성 및 객체 소유권 설정

일반 구성

버킷 이름

myawsbucket

버킷 이름은 전역에서 고유해야 하며 공백 또는 대문자를 포함할 수 없습니다. [버킷 이름 지정 규칙 참조](#)

AWS 리전

아시아 태평양(서울) ap-northeast-2

기존 버킷에서 설정 복사 - 선택 사항

다음 구성의 버킷 설정만 복사됩니다.

버킷 선택

객체 소유권 Info

다른 AWS 계정에서 이 버킷에 작성한 객체의 소유권 및 액세스 제어(ACL)의 사용을 제어합니다. 객체 소유권은 객체에 대한 액세스를 지정할 수 있는 사용자를 결정합니다.

ACL 비활성화됨(권장)

이 버킷의 모든 객체는 이 계정이 소유합니다. 이 버킷과 그 객체에 대한 액세스는 정책을 통해서만 지정됩니다.

ACL 활성화됨

이 버킷의 객체는 다른 AWS 계정에서 소유할 수 있습니다. 이 버킷 및 객체에 대한 액세스는 ACL을 사용하여 지정할 수 있습니다.

객체 소유권

버킷 소유자 선택

이 버킷에 작성된 새 객체가 bucket-owner-full-control 삽입 ACL을 지정하는 경우 새 객체는 버킷 소유자가 소유합니다. 그렇지 않은 경우 객체 라이터가 소유합니다.

객체 라이터

객체 라이터는 객체 소유자로 유지됩니다.

새 객체에 대해서만 객체 소유권을 적용하려면 버킷 정책이 객체 업로드에 bucket-owner-full-control 삽입 ACL을 요구하도록 지정해야 합니다. [자세히 알아보기](#)

ACL 활성화와 관련 향후 권한 변경 사항

2023년 4월부터는 S3 콘솔을 사용하여 버킷을 생성할 때 ACL을 활성화하기 위해 s3:PutBucketOwnershipControls 권한이 있어야 합니다. [자세히 알아보기](#)

## 2. 퍼블릭 액세스 설정

포팅 메뉴얼

5

## 이 버킷의 퍼블릭 액세스 차단 설정

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

### ☐ 모든 퍼블릭 액세스 차단

이 설정을 활성화하면 아래 4개의 설정을 모두 활성화한 것과 같습니다. 다음 설정 각각은 서로 독립적입니다.

#### ☐ 새 ACL(엑세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 새로 추가된 버킷 또는 객체에 적용되는 퍼블릭 액세스 권한을 차단하며, 기존 버킷 및 객체에 대한 새 퍼블릭 액세스 ACL 생성을 금지합니다. 이 설정은 ACL을 사용하여 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 권한을 변경하지 않습니다.

#### ☐ 임의의 ACL(엑세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 모든 ACL을 무시합니다.

#### ☐ 새 퍼블릭 버킷 또는 액세스 지점 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 새 버킷 및 액세스 지점 정책을 차단합니다. 이 설정은 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 정책을 변경하지 않습니다.

#### ☐ 임의의 퍼블릭 버킷 또는 액세스 지점 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 및 교차 계정 액세스 차단

S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 정책을 사용하는 버킷 또는 액세스 지점에 대한 퍼블릭 및 교차 계정 액세스를 무시합니다.



모든 퍼블릭 액세스 차단을 비활성화하면 이 버킷과 그 안에 포함된 객체가 퍼블릭 상태가 될 수 있습니다.

정적 웹 사이트 호스팅과 같은 구체적으로 확인된 사용 사례에서 퍼블릭 액세스가 필요한 경우가 아니면 모든 퍼블릭 액세스 차단을 활성화하는 것이 좋습니다.

☒ 현재 설정으로 인해 이 버킷과 그 안에 포함된 객체가 퍼블릭 상태가 될 수 있음을 알고 있습니다.

## 3. 버킷 정책 설정

생성된 버킷에 들어가서 권한을 누르고 버킷 정책을 설정한다.

(편집에 들어가서 아래의 검은색 부분에 입력해주면 된다 - 개개인별 설정 값)

버킷 정책

편집
삭제

JSON으로 작성된 버킷 정책은 버킷에 지정된 객체에 대한 액세스 권한을 제공합니다. 버킷 정책은 다른 계정이 소유한 객체에는 적용되지 않습니다. [자세히 알아보기](#)

{
}

복사

⇒ 나머지는 AWS S3의 기본 설정을 따른다.

## 2-4. WebServer

### ▼ 과정

※ 웹서버는 Nginx를 사용하고 EC2에 직접 설치한다.(도커사용x)

```
##설치
$ sudo apt install nginx

$ sudo apt-get install certbot
$ apt-get install python3-certbot-nginx

## 설정위치로 이동
$ cd /etc/nginx/sites-enabled

## 설정을 위해 vim 에디터 열기
$ sudo vim default.conf
```

```
server {
    server_name j8b206.p.ssafy.io;

    location / {
        proxy_pass http://localhost:3000;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_http_version 1.1;
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        #try_files $uri $uri/ =404;
    }

    #backend - api
    location /api/ {
        proxy_pass http://localhost:8080/;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_http_version 1.1;
    }

    #crawling - api
    location /crawling/ {
        proxy_pass http://localhost:8081/;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_http_version 1.1;
    }

    #fast - api
    location /fastapi/ {
        proxy_pass http://localhost:8000/;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_http_version 1.1;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/j8b206.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/j8b206.p.ssafy.io/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 ($host = j8b206.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80 default_server;
    listen [::]:80 default_server;
```

```
server_name j8b206.p.ssafy.io www.j8b206.p.ssafy.io;
return 404; # managed by Certbot

}
```

## 2-5. Jenkins

### ▼ 과정

※ docker in docker는 보안 문제가 있기 때문에 docker out of docker로 구성한다.

```
#Dockerfile
FROM jenkins/jenkins:lts
USER root

RUN apt-get update \
    && apt-get -y install lsb-release \
    && curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg \
    && echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb_release -cs) stable" >/etc/apt/sources.list.d/docker.list \
    && apt-get update \
    && apt-get -y install docker-ce docker-ce-cli containerd.io
RUN usermod -s /bin/bash docker jenkins

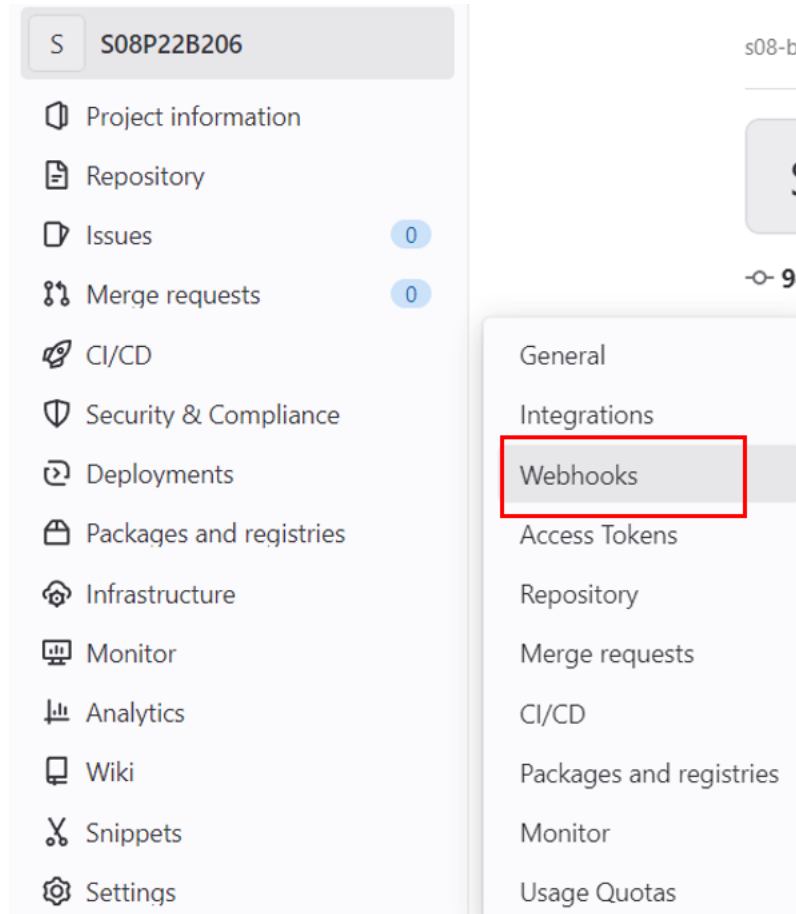
USER jenkins
```

```
#set volume
cd ~
mkdir jenkins

#docker run jenkins
sudo docker run -d -p 9090:8080 -it -v /home/ubuntu/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name ci_cd
```

- 웹훅설정





- 신호를 보낼 주소(젠킨스 파이프라인 구축시 생성되는 주소)  
(fe\_develop, be\_develop, crawling, data\_processing을 따로 만들)

### Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

http://example.com/trigger-ci.json

URL must be percent-encoded if it contains one or more special characters.

Secret token
Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

Trigger

☒ Push events

Branch name or wildcard pattern to trigger on (leave blank for all)

Push to the repository.

SSL verification
☒ Enable SSL verification

Add webhook

## 3. Backend

### 3-1. Spring 설정파일

#### ▼ 과정

application.yml

```
#application.yml
spring:
  profiles:
    default: develop
```

application-develop.yml

```
spring:
  ## 레디스 설정
  redis:
    host: <주소>
    port: <포트번호>
    key: <알람에 사용할 키>
  datasource:
    url: "jdbc:mysql://localhost:3306/mana_db?serverTimezone=UTC&characterEncoding=UTF-8"
    username: <계정>
    password: <비밀번호>
    driver-class-name: com.mysql.cj.jdbc.Driver

  servlet:
    multipart:
      max-request-size: 20MB
      max-file-size: 20MB
      enabled: true
  jpa:
    database:
    hibernate:
      ddl_auto: none #validate로 검증하려고 했는데, jpa2.1부터 지원안함
    properties:
      hibernate:
        "globally_quoted_identifiers": "true"
        format_sql: true
        show_sql: true #sys.out으로 sql 로그 남겨줌.
    database-platform: org.hibernate.dialect.MySQL8Dialect

# 스웨거 설정
mvc:
  pathmatch:
    matching-strategy: ANT_PATH_MATCHER
## 시큐리티 관련 설정 - 카카오 로그인
security:
  oauth2:
    client:
      registration:
        kakao:
          client-name: <클라이언트 이름>
          client-id: <받은 id>
          redirect-uri: <리다이렉트 uri>
          client-authentication-method: POST
          authorization-grant-type: authorization_code
          scope:
            - profile_nickname
            - account_email
            - profile_image
            - gender
            - age_range
      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

##aws 관련 설정.
cloud:
  aws:
    s3:
      bucket: <버킷명>
```

```

    directory:
      profile: user_profile/
  credentials:
    access-key: <access-key>
    secret-key: <secret-key>
  region:
    static: ap-northeast-2
    auto: false
  stack:
    auto: false
## jwt 관련
app:
  auth:
    token-secret: <token-secret>
    token-expiration-time: 86400000 # 토큰 만료일은 하루.
    refresh-token-secret: <refresh-token-secret>
    refresh-token-expiration-time: 604800000 #리프레시 토큰은 일주일
    redirect-page: <리다이렉트할 주소>
  oauth2:
    authorized-redirect-uris: <리다이렉트 할 주소>
## 네이버 api - keyword
openApi:
  naver:
    url: https://openapi.naver.com
    client-id: <네이버 제공 id>
    client-secret: <네이버 제공 키>
    keyword-rank:
      url: https://datalab.naver.com/shoppingInsight/getCategoryKeywordRank.naver
      cid: 50005566
      timeUnit: week
      header:
        user-agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Mobi
        referer: https://datalab.naver.com/shoppingInsight/sCategory.naver

```

#### application-deploy.yml

```

spring:
  ## 레디스 설정
  redis:
    host: <주소>
    port: <포트번호>
    key: <알람에 사용할 키>
  datasource:
    url: "jdbc:mysql://<RDS 도메인>:3306/mana_db?serverTimezone=UTC&characterEncoding=UTF-8"
    username: <계정>
    password: <비밀번호>
    driver-class-name: com.mysql.cj.jdbc.Driver

  servlet:
    multipart:
      max-request-size: 20MB
      max-file-size: 20MB
      enabled: true
  jpa:
    database:
    hibernate:
      ddl_auto: none #validate로 검증하려고 했는데, jpa2.1부터 지원안함
    properties:
      hibernate:
        "globally_quoted_identifiers": "true"
        format_sql: true
        show_sql: true #sys.out으로 sql 로그 남겨줌.
    database-platform: org.hibernate.dialect.MySQL8Dialect
# 스웨거 설정
mvc:
  pathmatch:
    matching-strategy: ANT_PATH_MATCHER
## 시큐리티 관련 설정 - 카카오 로그인
security:
  oauth2:
    client:
      registration:
        kakao:
          client-name: <클라이언트 이름>
          client-id: <받은 id>
          redirect-uri: <리다이렉트 uri>

```

```

    client-authentication-method: POST
    authorization-grant-type: authorization_code
    scope:
      - profile_nickname
      - account_email
      - profile_image
      - gender
      - age_range
    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
## 시큐리티 관련 설정 - 카카오 로그인
security:
  oauth2:
    client:
      registration:
        kakao:
          client-name: Kakao
          client-id: 759cbc8ee3b9d84f17b5d0473ae195b9
          redirect-uri: https://j8b206.p.ssafy.io/api/login/oauth2/code/kakao
          client-authentication-method: POST
          authorization-grant-type: authorization_code
          scope:
            - profile_nickname
            - account_email
            - profile_image
            - gender
            - age_range
        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

##aws 관련 설정.
cloud:
  aws:
    s3:
      bucket: <버킷명>
      directory:
        profile: user_profile/
    credentials:
      access-key: <access-key>
      secret-key: <secret-key>
    region:
      static: ap-northeast-2
      auto: false
    stack:
      auto: false
## jwt 관련
app:
  auth:
    token-secret: <token-secret>
    token-expiration-time: 86400000 # 토큰 만료일은 하루.
    refresh-token-secret: <refresh-token-secret>
    refresh-token-expiration-time: 604800000 #리프레시 토큰은 일주일
    redirect-page: <리다이렉트할 주소>
  oauth2:
    authorized-redirect-uris: <리다이렉트 할 주소>

## 네이버 api - keyword
openApi:
  naver:
    url: https://openapi.naver.com
    client-id: <네이버 제공 id>
    client-secret: <네이버 제공 키>
    keyword-rank:
      url: https://datalab.naver.com/shoppingInsight/getCategoryKeywordRank.naver
      cid: 50005566
      timeUnit: week
    header:
      user-agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Mobile
      referer: https://datalab.naver.com/shoppingInsight/sCategory.naver

```

## 3-2 Dockerfile

### ▼ 과정

빌드 후 배포할 도커파일작성

```
FROM openjdk:11-jdk

ENV APP_HOME=/usr/app

WORKDIR $APP_HOME

COPY ./manamana/build/libs/*.jar ./application.jar

COPY ./resources ./resources

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "-Dspring.config.location=resources/application.yml,resources/application-deploy.yml", "-Dspring.profiles.act
```

## 3-3 Jenkinsfile

### ▼ 과정

파이프라인 설정

빌드 - 이전 컨테이너 및 이미지 삭제 - 도커파일 작성 - 도커 실행

```
pipeline {
    agent any
    stages {
        stage("Set Variable") {
            steps {
                script {
                    IMAGE_NAME = "sh80165/manamana-springboot"
                    IMAGE_STORAGE = "https://registry.hub.docker.com"
                    IMAGE_STORAGE_CREDENTIAL = "docker-auth"
                    SPRING_BUILD_PATH = "./backend/manamana"
                    APPLICATION_YML_PATH = "/var/jenkins_home/workspace"
                    CONTAINER_NAME = "manamana-api"
                    PROJECT_DIR = "spring-boot/backend/"
                    DOCKER_FILE_PATH = "backend/"
                }
            }
        }
        stage("Clean Build Test") {
            steps {
                dir("${SPRING_BUILD_PATH}") {
                    sh "pwd"
                    sh "chmod +x gradlew"
                    sh "./gradlew clean build -x test"
                    sh "ls -al ./build"
                }
            }
        }
        stage("Copy Application.yml"){
            steps{
                dir("${APPLICATION_YML_PATH}") {
                    sh "pwd"
                    sh "cp -r -f resources ${PROJECT_DIR}"
                }
            }
        }
        stage("Build Container Image") {
            steps {
                dir("${DOCKER_FILE_PATH}") {
                    script {
                        image = docker.build("${IMAGE_NAME}")
                    }
                }
            }
        }
    }
}
```

```

    }

    stage("Push Container Image") {
        steps {
            script {
                docker.withRegistry("", "${IMAGE_STORAGE_CREDENTIAL}") {
                    image.push("latest")
                }
            }
        }
    }

    stage("Server Run") {
        steps {
            script {
                // //컨테이너 확인 후 정지
                sh "docker ps -f name=${CONTAINER_NAME} -q | xargs --no-run-if-empty docker container stop"

                // //컨테이너 삭제
                sh "docker container ls -a -f name=${CONTAINER_NAME} -q | xargs -r docker container rm"

                //기존 이미지 삭제
                sh "docker images sh80165/manamana-springboot -q | xargs -r docker rmi -f"

                //컨테이너 확인
                sh "docker ps -a"

                // 최신 이미지 PULL
                sh "docker pull ${IMAGE_NAME}:latest"

                // 이미지 확인
                sh "docker images"

                // 최신 이미지 RUN
                sh "docker run -d --name ${CONTAINER_NAME} --link recommend-api -p 8080:8080 ${IMAGE_NAME}:latest"

                // 컨테이너 확인 - 로그 확인용
                sh "docker ps -a"
            }
        }
    }
}
}
}
}
}

```

## 4. Frontend

### 3-1 Dockerfile

#### ▼ 과정

npm 대신 yarn을 이용해서 install 속도를 올림.

```

# 노드 버전 지정
FROM node:16-alpine

LABEL email="lsh80165@gmail.com"

ENV APP_HOME=/usr/app

WORKDIR $APP_HOME

COPY . .

RUN yarn cache clean

# RUN yarn add @mui/icons-material --network-timeout 500000

RUN yarn install

RUN npm run build
#어떤 포트에서 listen할지
EXPOSE 3000

```

```
ENTRYPOINT [ "npm", "run", "start" ]
```

### 3-3 Jenkinsfile

#### ▼ 과정

```
pipeline {
    agent any

    stages {
        stage("Set Variable") {
            steps {
                script {
                    IMAGE_NAME = "sh80165/manamana-frontend"
                    IMAGE_STORAGE = "https://registry.hub.docker.com"
                    IMAGE_STORAGE_CREDENTIAL = "docker-auth"
                    NODE_BUILD_PATH = "./build"
                    APPLICATION_ENV_PATH = "/var/jenkins_home/workspace"
                    ESLINTIGNORE_PATH = "/var/jenkins_home/workspace/manamana-frontend"
                    CONTAINER_NAME = "manamana-fronted"
                    PROJECT_DIR = "frontend"
                    DOCKER_FILE_PATH = "frontend/"
                }
            }
        }

        stage("Node install & build") {
            steps {
                dir("${ESLINTIGNORE_PATH}") {
                    sh "pwd"
                    sh "cp -f eslintignore frontend/eslintignore"
                }
            }
        }

        stage("env copy") {
            steps {
                dir("${APPLICATION_ENV_PATH}") {
                    sh "pwd"
                    sh "cp -f fronted_env/.env manamana-frontend/frontend/.env"
                }
            }
        }

        stage("Clean Container&Image") {
            steps {
                script{
                    // //컨테이너 확인 후 정지
                    sh "docker ps -f name=${CONTAINER_NAME} -q | xargs --no-run-if-empty docker container stop"

                    // //컨테이너 삭제
                    sh "docker container ls -a -f name=${CONTAINER_NAME} -q | xargs -r docker container rm"

                    //기존 이미지 삭제
                    sh "docker images ${IMAGE_NAME} -q | xargs -r docker rmi -f"
                }
            }
        }

        stage("Build Container Image") {
            steps {
                dir("${DOCKER_FILE_PATH}") {
                    script {
                        sh "pwd"
                        image = docker.build("${IMAGE_NAME}")
                    }
                }
            }
        }

        stage("Push Container Image") {
            steps {
                script {
                    docker.withRegistry("", "${IMAGE_STORAGE_CREDENTIAL}") {
                        image.push("latest")
                    }
                }
            }
        }
    }
}
```

```

    }
  }
}

stage("Server Run") {
  steps {
    script{

      //컨테이너 확인
      sh "docker ps -a"

      // 이미지 확인
      sh "docker images"

      // 최신 이미지 RUN
      sh "docker run -d --name ${CONTAINER_NAME} -p 3000:3000 ${IMAGE_NAME}:latest"

      // 컨테이너 확인 - 로그 확인용
      sh "docker ps -a"
    }
  }
}
}
}
}

```