포팅 메뉴얼

▼ **목차**

1. 개발 환경
2. 환경 세팅
    1) Docker 설치
    2) MariaDB 설치 및 설정
    3) Redis 설치 및 설정
3. 분산처리
    1) NASA로 부터 광공해 위성 데이터 획득
    2) 유효한 X,Y 날씨 격자 획득
    3) 날씨 크롤링
    4) 스파크 분산 처리
    5) REDIS 데이터 삽입
4. 프론트엔드 빌드 및 배포
    1) 환경변수파일 추가
    2) Docker 이미지 생성
    3) 배포
    추가) 프론트엔드 배포관련 설정 파일 설명
5. 백엔드 빌드 및 배포
    1) application.yml 파일 작성 및 추가
    2) Spring boot 프로젝트 빌드
    3) Docker 이미지 생성
    4) 배포
    추가) 백엔드 배포관련 설정 파일 설명

1. 개발 환경

소통채널

- Mattermost
- Webex
- Notion
- Figjam

데이터베이스

- Redis : 7.0.10
- MariaDB : 10.11.2

이슈관리

- Jira

UI/UX

- Figma

형상관리

- GitLab

IDE

- IntelliJ IDEA : 2022.3.1

기타 툴

- Postman : v10.12.10
- Visual Studio Code : 1.74.2

서버

- AWS EC2
- Ubuntu : 20.04 LTS
- Docker : 23.0.1

---

2. 환경 세팅

1) Docker 설치

- Ubuntu 20.04 기준, 아래의 가이드를 따라 Docker를 설치한다.

Install Docker Engine on Ubuntu

Jumpstart your client-side server applications with Docker Engine on Ubuntu. This guide details prerequisites and multiple methods to install.

https://docs.docker.com/engine/install/ubuntu/

2) MariaDB 설치 및 설정

1. MariaDB 이미지를 내려받는다.

```
docker pull mariadb
```

2. 컨테이너 실행

```
docker run -p 3206:3206 --name mariadb -e MARIADB_ROOT_PASSWORD={비밀번호} -d mariad
b
```

3. DB 사용자 추가

   1) mariadb 컨테이너 접속

```
docker exec -it mariadb /bin/bash
```

2) 로그인, 아래의 커맨드를 입력하고 컨테이너를 생성할 때 입력한 비밀번호로 로그인 한다.

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

3) 모든 DB, 테이블에 접속 가능한 사용자 생성

```
use mysql
CREATE USER '{사용자 이름}'@'%' IDENTIFIED BY '{비밀번호}';
GRANT ALL PRIVILEGES ON *.* TO '{사용자 이름}'@'%';
FLUSH PRIVILEGES;
```

3) Redis 설치 및 설정

1. redis.conf 파일 생성

```
requirepass yourpassword
```

2. DockerFile 작성

```
FROM redis
COPY redis.conf /usr/local/etc/redis/redis.conf
CMD ["redis-server", "/usr/local/etc/redis/redis.conf"]
```

3. 도커 이미지 빌드

```
docker build -t myredis .
```

4. 컨테이너 실행

```
docker run -d -p 6400:6379 -v /path/to/redis.conf:/usr/local/etc/redis/redis.conf
 --name myredis redis redis-server /usr/local/etc/redis/redis.conf
```

3. 분산처리

1) NASA로 부터 광공해 위성 데이터 획득

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/8531227d-01c7-4d70-8a09-c9900dd93e7f/ReadMe.md

마크다운의 안내에 따라 ImageJ 툴을 사용하여, 위성 데이터를 grayscale로 추출한다.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/98e8fda6-e2ef-4677-aea7-f5b84a1bc31f/light_pollution_pre_processor.py

광공해 데이터를 타일로 요약 처리하여, 여러 스케일의 파일로 나눈다.

2) 유효한 X,Y 날씨 격자 획득

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/66c70000-d869-4958-9a29-7067555465ee/koreanGrid.py

- 기상청의 예보 기준 구역 : 5km x 5km 격자
- 행정동과 대응이 되는 유효한 격자만 획득하여 아래의 csv파일로 저장한다.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/679f0cc8-6609-4afb-ab1b-54cfddd15c05/weather_geo_info.csv

3) 날씨 크롤링

- 상기 파이썬 스크립트로 획득한 weather_geo_info.csv를 토대로 날씨를 크롤링 한다.
- 결과 : 남한 육지 상에 유효한 전국 날씨 데이터 획득

  https://s3-us-west-2.amazonaws.com/secure.notion-static.com/2453ca83-a9f8-4fdb-a4d9-980731ff7a34/weather_scraper.py

- crontab을 활용하여 서버에서 주기적인 크롤링을 수행한다.
- 주의 : 낮 시간대와 같이 트래픽이 몰릴 때는 정상적으로 크롤링이 안됨

4) 스파크 분산 처리

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/d8cbe0d1-3b24-4301-99ba-140ece446c36/grid.py

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/76cca434-a524-495d-bae8-460fbed97275/light_weather_analyzer.py

```
spark-submit light_weather_anyalyzer.py
```

명령어를 수행하여, 광공해 위성 데이터와 날씨 데이터를 결합한다.

5) REDIS 데이터 삽입

> https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4e0fece3-53f8-4858-9817-ad77d9b0bce3/redis_data_send.py

스크립트를 실행하여 redis 컨테이너에 처리된 데이터를 삽입한다.

---

4. 프론트엔드 빌드 및 배포

- 환경변수 파일은 보안을 위해 git 원격저장소에 업로드 하지 않음.
- 따라서 **.env.production.local** , **.env.development.local** 파일을 추가해야함.

1) 환경변수파일 추가

- /frontend/**.env.production.local**

```
REACT_APP_PROFILE_INFO=배포
REACT_APP_NAVER_MAP_CLIENT_ID={네이버맵 클라이언트 ID}
REACT_APP_API_SERVER_URL=https://{백엔드서버도메인}/api
```

- /frontend/**.env.development.local**

```
REACT_APP_PROFILE_INFO=로컬
REACT_APP_NAVER_MAP_CLIENT_ID={네이버맵 클라이언트 ID}
REACT_APP_API_SERVER_URL=http://localhost:8090/api
```

2) Docker 이미지 생성

```
docker build -t starry-night-fe -f frontend/Dockerfile .
```

3) 배포

- 이미 실행되고 있는 starry-night-fe 컨테이너가 있다면 삭제한다.

```
docker rm -f starry-night-fe
```

- 컨테이너를 생성한다.

```
docker run -d -p 3080:80 --name starry-night-dev-fe starry-night-fe
```

- 사용한 옵션
  - `-d`:
    - 컨테이너를 백그라운드에서 실행
  - `-p 3080:80`:
    - 호스트 머신의 `3080` 포트와 컨테이너 내부의 `80` 포트 매핑
  - `-name starry-night-dev-fe`:
    - 컨테이너 이름을 `starry-night-dev-fe`로 설정

추가) 프론트엔드 배포관련 설정 파일 설명

- **Dockerfile**

```
# build-stage
FROM node:lts-alpine as build-stage
# /app 디렉토리를 작업 디렉토리로 설정
WORKDIR /app
# ../frontend 내부 파일을 작업 디렉토리로 복사
COPY ../frontend .
# 의존성 설치
RUN npm install --force
# 프로젝트 빌드
RUN npm run build

# production-stage
FROM nginx:stable-alpine as production-stage
# build-stage에서 빌드된 정적파일을 복사
COPY --from=build-stage /app/build /usr/share/nginx/html
# nginx.conf 파일 복사
COPY ../frontend/nginx.conf /etc/nginx/conf.d/default.conf
# 80포트 개방
EXPOSE 80
# nginx 실행
CMD ["nginx", "-g", "daemon off;"]
```

- **nginx.conf**

```
server {
  listen       80; # 80 포트로 들어오는 요청 처리
  server_name  localhost; # 해당 설정이 적용될 도메인
  location / { # 클라이언트의 요청 URL과 Nginx 서버의 파일 경로 매핑
    root   /usr/share/nginx/html; # 해당 location에서 요청한 파일의 루트 디렉토리
    index  index.html; # 디렉토리에서 기본으로 사용할 인덱스 파일
    try_files $uri $uri/ /index.html; # 요청이 정적 파일인 경우 해당 파일 반환, 만약 해당 파일이
 존재하지 않으면 /index.html 파일 반환
  }
}
```

---

5. 백엔드 빌드 및 배포

- 환경변수 파일은 보안을 위해 git 원격저장소에 업로드 하지 않음.
- 따라서 application.yml 파일을 추가해야함.

1) application.yml 파일 작성 및 추가

- 위치 : /backend/src/main/resources/application.yml

```
spring:
  profiles:
    active: dev # 기본 활성 프로파일을 "dev"로 설정
  redis:
    host: {호스트} # Redis 호스트
    port: {포트} # Redis 포트 번호
    password: {암호} # Redis 암호
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver # MariaDB JDBC 드라이버 클래스 이름
    url: jdbc:mariadb://{호스트:포트}/starrynight # MariaDB 데이터베이스 URL
    username: {사용자} # MariaDB 사용자 이름
    password: {암호} # MariaDB 암호
  jpa:
    hibernate:
      ddl-auto: update # Hibernate DDL 자동 생성 전략 설정
    show-sql: true # SQL 쿼리 출력 여부 설정

  security:
    oauth2:
      client:
        registration:
```

```
          kakao:
            client-id: {카카오 클라이언트 ID} # 카카오 클라이언트 ID
            client-secret: {카카오 클라이언트 시크릿} # 카카오 클라이언트 시크릿
            authorization-grant-type: authorization_code # 카카오 인증 유형
            client-authentication-method: POST # 클라이언트 인증 방식
            client-name: Kakao # 클라이언트 이름
            scope: # 클라이언트 요청 스코프
              - profile_nickname
              - profile_image
          google:
            client-id: {구글 클라이언트 ID} # 구글 클라이언트 ID
            client-secret: {구글 클라이언트 시크릿} # 구글 클라이언트 시크릿
            scope:
              - email
              - profile

        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
  servlet:
    multipart:
      max-file-size: 50MB # 업로드 파일 최대 크기 설정
      max-request-size: 50MB # 요청 최대 크기 설정

jwt:
  secret: {JWT 시크릿 키} # JWT 시크릿 키
  issuer: {JWT 발급자} # JWT 발급자

app:
  auth:
    access-token-name: access_token # 액세스 토큰 이름

file:
  dir: /usr/app/starry-night-dev/ # 파일 저장 경로

---
# prod 프로파일 설정
spring:
  config:
    activate:
      on-profile: prod

  security:
    oauth2:
      client:
        registration:
          kakao:
            redirect-uri: https://{도메인}/api/login/oauth2/code/kakao
          google:
            redirect-uri: https://{도메인}/api/login/oauth2/code/google

server:
  port: 18090 # 서버 포트 번호 설정
  servlet:
    context-path: /api # 서블릿 컨텍스트 경로 설정
```

```
app:
  oauth2:
    authorizedRedirectUri: "http://{도메인}" # 카카오 로그인 후 리다이렉션 URI
  uri:
    scheme: https # URI 스키마 설정
    domain: {도메인} # 도메인 설정
    port: 443 # 포트 설정

---
# dev 프로파일 설정
spring:
  config:
    activate:
      on-profile: dev

  datasource:
    driver-class-name: org.mariadb.jdbc.Driver # MariaDB JDBC 드라이버 클래스 이름
    url: jdbc:mariadb://localhost:3206/starrynight # MariaDB 데이터베이스 URL
    username: {사용자} # MariaDB 사용자 이름
    password: {암호} # MariaDB 암호

  security:
    oauth2:
      client:
        registration:
          kakao:
            redirect-uri: http://localhost:8090/api/login/oauth2/code/kakao
          google:
            redirect-uri: http://localhost:8090/api/login/oauth2/code/google

server:
  port: 8090
  servlet:
    context-path: /api

app:
  oauth2:
    authorizedRedirectUri: "http://localhost:3000"
  uri:
    scheme: http
    domain: localhost
    port: 8090

file:
  dir: C:\files\starry-night\
```

2) Spring boot 프로젝트 빌드

```
./gradlew clean build
```

3) Docker 이미지 생성

```
docker build -t starry-night-server ./backend
```

4) 배포

- 이미 실행되고 있는 starry-night-server 컨테이너가 있다면 삭제한다.

```
docker rm -f starry-night-server
```

- 컨테이너를 생성한다.

```
docker run -d -p 18090:18090 -v /home/ubuntu/app/starry-night-dev:/usr/app/starry-nigh
t-dev -e TZ=Asia/Seoul -e USE_PROFILE=prod --name starry-night-server starry-night-ser
ver
```

- 사용한 옵션
  - `-d` :
    - 컨테이너를 백그라운드에서 실행
  - `-p 18090:18090` :
    - 호스트 머신의 `18090` 포트와 컨테이너 내부의 `18090` 포트를 매핑
  - `-v /home/ubuntu/app/starry-night-dev:/usr/app/starry-night-dev` :
    - 호스트 머신의 `/home/ubuntu/app/starry-night-dev` 디렉토리를 컨테이너 내부의 `/usr/app/starry-night-dev` 디렉토리로 마운트
  - `-e TZ=Asia/Seoul` :
    - 컨테이너 내부의 시간대를 `Asia/Seoul`로 설정
  - `-e USE_PROFILE=prod` :
    - `USE_PROFILE` 환경 변수를 `prod`로 설정
  - `--name starry-night-server` :
    - 컨테이너 이름을 `starry-night-server`로 설정

추가) 백엔드 배포관련 설정 파일 설명

- **Dockerfile**

```
FROM adoptopenjdk/openjdk11
WORKDIR /usr/app
COPY build/libs/*.jar starry-night-server.jar
ENV USE_PROFILE local
CMD ["java","-Dspring.profiles.active=${USE_PROFILE}", "-jar","starry-night-server.ja
r"]
```