[다음:날] 포팅 매뉴얼

목차

1. 개발 환경
2. 기술 스택
3. 환경 변수 및 설정
4. 배포 시 특이사항
5. 외부 서비스 정보

1. 개발 환경

A. 공통

i. Jira

ii. Notion

iii. Mattermost

iv. Gerrit

v. Git

vi. GitLab

vii. Postman

viii. Figma

B. 프론트엔드

i. React 18.2.0

ii. TypeScript 4.9.5

iii. Tailwild CSS 3.4.1

iv. Axios 1.6.7

v. Oauth 2.0

vi. Visual Studio Code 1.85.1

C. 백엔드

i. Java 17.0.9

ii. Intellij 2023.3.2

iii. Spring Boot 3.2.3

iv. Spring Data JPA 3.2.3

v. Spring Security 6.2.2

vi. JWT 0.12.3

vii. MySQL 8.3.0

viii. Redis 7.2.4

ix. Python 3.8.10

x. FastAPI 0.110.0

xi. Pycharm 2023.3.2

xii. Kobert 0.2.3

xiii. Selenium 4.17.2

D. 인프라

i. EC2

ii. S3

iii. Ubuntu 20.04.6

iv. Jenkins 2.448

v. Nginx 1.25.4

vi. Certbot 0.40.0

vii. TLS 1.3

viii. Docker 25.0.4

ix. docker compose 1.29.2

2. 기술 스택

A. Development

i. Java & Spring Boot

ii. Python & FastAPI

iii. TypeScript & React

B. Deployment

i. GitLab & Jenkins Webhook

ii. docker compose & Docker

iii. Nginx Reverse Proxy

iv. Nginx & React Web Server

v. Spring Boot & FastAPI End Server

C. Sign up & Sign in

i. Java & Spring Boot

ii. TypeScript & React

iii. KaKao Oauth

iv. Spring Security & JWT & Redis

D. EC2 Setting

i. Ubuntu

E. Nginx Setting

i. Docker

ii. Certbot & SSL/TLS & HTTPS

F. MySQL 접속 명령어

i. EC2 에서 bash 를 이용해 MySQL Docker 컨테이너에 접속

docker exec –it <컨테이너명> bash

ii. MySQL 계정 로그인 (root 계정 로그인)

1. docker exec –it <컨테이너명> bash
2. mysql –uroot –p
3. 패스워드 입력

G. Redis 접속 명령어

i. EC2 에서 bash 를 이용해 Redis Docker 컨테이너에 접속

docker exec –it <컨테이너명> bash

ii. EC2 에서 Redis Client 접속 (패스워드 미적용 시)

docker exec –it <컨테이너명> redis-cli

iii. EC2 에서 Redis Client 접속 (패스워드 적용 시)

docker xec –it <컨테이너명> redis-cli –a <패스워드>

3. 환경 변수 및 설정

A. Spring Boot

i. application.yml, application-dev.yml, application-s3.yml

B. FastAPI

i. requirements.txt

C. 사진

i. application.yml

```yaml
spring:
  config:
    import:
      - classpath:/application-dev.yml
      - classpath:/application-s3.yml
```

ii. application-dev.yml

```yaml
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://${EC2_ACCESS_IP}:3306/daumnal?serverTimezone=Asia/Seoul&useUnicode=true&characterEncoding=utf8&useLegacyDatetimeCode=false
    username: root
    password: ${MYSQL_PW}
  jpa:
    database: mysql
    database-platform: org.hibernate.dialect.MySQLDialect
    hibernate:
      ddl-auto: none
      naming:
        physical-strategy: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
    properties:
      globally_quoted_identifiers: true
      hibernate:
        format_sql: true
        show_sql: true
  jwt:
    secret: ${JWT_SECRET}
    live:
      access: 3600000 # access token 기한 1시간
      refresh: 604800000 # refresh token 기한 일주일

  cache:
    type: redis

  data:
    redis:
      host: ${EC2_ACCESS_IP}
      port: 6379

  servlet:
    multipart:
      max-file-size: 3MB
      max-request-size: 3MB
```

iii. application-s3.yml

```yaml
cloud:
  aws:
    s3:
      bucket: daumnal
    credentials:
      accessKey: ${S3_ACCESSKEY}
      secretKey: ${S3_SECRETKEY}
    region:
      static: ap-northeast-2
      auto: false
    stack:
      auto: false
```

iv. requirements.txt

```
mxnet==1.6.0
gluonnlp==0.10.0
pandas==2.0.3
sentencepiece==0.2.0
transformers==4.38.2
torch==2.2.1
numpy==1.23.1
openpyxl==3.1.2
scikit-learn==1.3.2
fastapi==0.110.0
pydantic==2.6.3
uvicorn==0.27.1
httpx==0.27.0
SQLAlchemy==2.0.28
PyMySQL==1.1.0
selenium==4.17.2
undetected-chromedriver==3.5.5
requests==2.31.0
```

4. 배포 시 특이사항

A. EC2 서버에 Docker 로 MySQL 설치

i. Dockerfile 작성

ii. docker run --name <컨테이너명> -d -p <로컬 포트>:<도커 포트> -e TZ=<지역명> -v <볼륨명>:<컨테이너 디렉토리 경로> --network <네트워크명> <이미지명>

B. EC2 서버에 Docker 로 Redis 설치

i. Dockerfile 작성

ii. docker run --name <컨테이너명> -d -p <로컬 포트>:<도커 포트> -e TZ=<지역명> -v <볼륨명>:<컨테이너 디렉토리 경로> --network <네트워크명> <이미지명>

C. EC2 서버에 Docker 로 Spring Boot 배포

i. Dockerfile 작성

ii. docker run --name <컨테이너명> -d --network <네트워크명> -e TZ=<지역명> <이미지명>

D. EC2 서버에 Docker 로 FastAPI 배포

i. Dockerfile 작성

ii. docker run --name <컨테이너명> -d --network <네트워크명> -e TZ=<지역명> <이미지명>

E. EC2 서버에 Docker 로 React 설치

i. Dockerfile 작성

ii. docker run --name <컨테이너명> -d -v <볼륨명>:<컨테이너 디렉토리 경로> -e TZ=<지역명> --network <네트워크명> <이미지명>

F. EC2 서버에 Docker 로 Jenkins 설치

i. Dockerfile 작성

ii. docker run --name <컨테이너명> -d -p <로컬 포트>:<도커 포트> -v <볼륨명>:<컨테이너 디렉토리 경로> -e TZ=<지역명> -u root <이미지명>

G. EC2 서버에 Docker 로 Nginx 배포

i. Dockerfile 작성

ii. docker run --name <컨테이너명> -d -p <로컬 포트>:<도커 포트> -v <볼륨명>:<컨테이너 디렉토리 경로> -e TZ=<지역명> --network <네트워크명> <이미지명>

H. EC2 서버에 HTTPS 설치

i. SSL 인증서 발급 (Webroot 방식)

1. sudo cerbot certonly --webroot --agree-tos -m <이메일> -w <EC2 내부 디렉토리 경로> -d <도메인명>
2. Certificate 와 Key 가 .pem 파일 형태로 EC2 에 저장

ii. Nginx 와 HTTPS 연결

1. /etc/nginx/conf.d 경로 내의 default.conf 파일 수정

```
map $http_origin $allowed_origin {
    default "";
    "http://localhost:3000" $http_origin;
    "https://<도메인명>" $http_origin;
}

server {
    listen 80;
    listen [::]:80;
    server_name <도메인명>;
    server_tokens off;

    return 308 https://$host$request_uri;
}

server {
    listen 4000 ssl;
    listen [::]:4000 ssl;
    server_name <도메인명>;
    server_tokens off;

    ssl_certificate /etc/letsencrypt/live/<도메인명>/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/<도메인명>/privkey.pem;

    location /api {
        proxy_set_header Host $http_host;
        proxy_pass http://<컨테이너명>:<도커 포트>;
    }

    location / {
        proxy_pass http://<컨테이너명>:<도커 포트>;
    }
}
```

2. docker exec –it <nginx 컨테이너명> bash
3. nginx –s reload

I. EC2 서버에 전체 서비스를 docker compose 로 배포

i. docker compose 작성

```yaml
version: "3.8"
services:
  mysql:
    container_name: daumnal-mysql
    image: mysql-image
    build:
      context: daumnal
      dockerfile: Dockerfile
    ports:
      - 3306:3306
    environment:
      - TZ=Asia/Seoul
    volumes:
      - mysql-volume:/var/lib/mysql
    networks:
      - daumnal-network
    restart: always
  redis:
    container_name: daumnal-redis
    image: redis-image
    build:
      context: daumnal
      dockerfile: Dockerfile
    ports:
      - 6379:6379
    environment:
      - TZ=Asia/Seoul
    volumes:
      - redis-conf-volume:/usr/local/etc/redis
      - redis-data-volume:/data
    networks:
      - daumnal-network
    restart: always
```

```yaml
spring:
  container_name: daumnal-spring
  image: spring-image
  build:
    context: daumnal
    dockerfile: Dockerfile
  ports:
    - 8080:8080
  depends_on:
    - mysql
    - redis
  environment:
    - TZ=Asia/Seoul
  networks:
    - daumnal-network
  restart: always
fastapi:
  container_name: daumnal-fastapi
  image: fastapi-image
  build:
    context: daumnal
    dockerfile: Dockerfile
  ports:
    - 8000:8000
  depends_on:
    - mysql
    - redis
  environment:
    - TZ=Asia/Seoul
  networks:
    - daumnal-network
  restart: always
```

```
  nginx:
    container_name: daumnal-nginx
    image: nginx-image
    build:
      context: daumnal
      dockerfile: Dockerfile
    ports:
      - 80:80
      - 443:443
    depends_on:
      - spring
      - fastapi
      - react
    volumes:
      - nginx-volume:/etc/nginx
      - /etc/letsencrypt:/etc/letsencrypt
      - /var/www/certbot:/var/www/certbot
    networks:
      - daumnal-network
    restart: always
  react:
    container_name: daumnal-react
    image: react-image
    build:
      context: daumnal
      dockerfile: Dockerfile
    environment:
      - TZ=Asia/Seoul
    volumes:
      - react-volume:/etc/nginx
    networks:
      - daumnal-network
```

```
volumes:
  mysql-volume:
    external: true
  redis-conf-volume:
    external: true
  redis-data-volume:
    external: true
  nginx-volume:
    external: true
  react-volume:
    external: true
networks:
  daumnal-network:
    external: true
```

ii. docker compose up –d

5. 외부 서비스 정보

A. 카카오 소셜 로그인

i. 카카오 로그인 API 문서

https://developers.kakao.com/docs/latest/ko/kakaologin/rest-api

ii. 카카오 개발자

https://developers.kakao.com/