Whirlyric 포팅매뉴얼

프로젝트 기술 스택

Infra	버전
AWS EC2	Ubuntu 20.04.6 LTS
Docker	26.0.2
Nginx	1.25.5
Jenkins	2.440.3
MySQL	8.3.0

- Spring Boot 종속성 관리에 따른 경우 버전 명시하지 않음
- https://docs.spring.io/spring-boot/docs/current/reference/html/dependency-versions.html#appendix.dependency-versions.properties -자세한 버전정보 참조

Backend	버전
Java	17.0.10 (Zulu)
Gradle	8.7
Spring Boot	3.2.4
spring-boot-starter-data-jpa	
spring-boot-starter-security	
querydsl-jpa	5.0.0:jakarta
spring-boot-starter-data-redis	
Lombok	1.18.30
Springdoc	2.0.2
spring-boot-starter-webflux	
json-simple:json-simple	1.1.1
javax.xml.bind:jaxb-api	2.3.0
spring-boot-starter-oauth2-client	

Frontend	버전
Node.js	v20.10.0
React	18.2.0
yarn	1.22.21
tailwindCSS	3.4.3

axios	1.6.8
react-router-dom	6.23.0
vite	5.2.0
typescript	5.2.2
autoprefixer	10.4.19
eslint	8.57.0

EC2 세팅

- 타임존 서울로 변경

```
sudo timedatectl set-timezone Asia/Seoul
```

- 스왑영역 할당

```
free -h     // 확인
sudo fallocate -l 4G /swapfile  - 4기가 할당
sudo chmod 600 /swapfile 권한수정
sudo mkswap /swapfile 파일생성
sudo swapon /swapfile 활성화
sudo echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab 재부팅해도 스왑
```

- 카카오 미러서버 repo 설정

```
sudo vi /etc/apt/sources.list
:%s/{바꿀주소}/mirror.kakao.com/c  c-> 인터렉티브 옵션
```

- 도커설치(공식 설치 페이지 설명 -> EC2 리눅스 버전에 맞게 바꿔서 사용)

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyr

# 도커 repo 추가
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
```

```
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

도커 최신버전으로 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plug

# 일반유저에게 도커 권한 주기
sudo usermod -aG docker [username] (도커 그룹에 사용자 추가)
도커 재시작
sudo service docker restart
유저도 재접속 -> sudo 안쳐도 됨
```

- 필요한 유틸 설치 ex) 트리

```
sudo apt-get install tree
```

- SSL 설정

```
* ZeroSSL 사용

1) 무료 도메인 등록
2) CNAME 방식으로 CA 인증서 발급
3) 리눅스 서버로 복사
3) OPENSSL -> crt를 pem형식으로 변경(crt파일 내용이 pem형식에 맞게 되어있으면 안해도 됨)
4) nginx 설정에 추가
5) 도커 볼륨에 추가
6) 컨테이너 안으로 들어가서 원하는 경로로 잘 갔는지 확인
```

- Jenkins 설정

```
- nginx띄우고, 젠킨스 띄우고, 대시보드 접근 -> admin password 입력
- getting started -> 주요 플로그인 모두 설치 선택
- 이후 대시보드 접속하고 파이프라인 작성에 필요한 플러그인 모두 설치

docker-compose.yml

networks:
  whirrlyric:
    name: whirrlyric
    external: true
```

```
services:
  jenkins:
    image: jenkins/jenkins:lts-jdk17
    container_name: jenkins
    environment:
      - TZ=Asia/Seoul
      - JENKINS_OPTS="--prefix=/jenkins"
      - JAVA_OPTS="-Dfile.encoding=UTF-8"
    user: root
    privileged: true
    ports:
      - 9999:8080
      - 50000:50000
    volumes:
      - ./config:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - whirrlyric
```

- DB 설정(MySQL)

```
- mysql 8.xx 이미지로 띄우기
- 환경변수에 root, user, password 등 지정
- 스프링 컨테이너와 동일한 도커 네트워크로 설정하고, 개발이 완전히 끝난 상황에선 외부경로를 통
-> 마지막 배포 때 application.yml datasource에서 도메인이 아니라 db 컨테이너 이름으로 바

docker-compose.yml

networks:
  whirrlyric:
    name: whirrlyric
    external: true
services:
  db:
    image: mysql:8
    container_name: mysql
    ports:
      - 3306:3306
    volumes:
      - ./conf.d:/etc/mysql/conf.d
      - ./data:/var/lib/mysql
      - ./initdb.d:/docker-entrypoint-initdb.d
    networks:
      - whirrlyric
```

```
    environment:
      TZ: Asia/Seoul
      MYSQL_ROOT_PASSWORD: ***** # 루트 비밀번호 설정
      MYSQL_DATABASE: ***********       # 생성할 데이터베이스 이름
      MYSQL_USER: *****               # 생성할 사용자 이름
      MYSQL_PASSWORD: *************       # 사용자 비밀번호
    restart: always
```

- redis 설정

```
docker-compose.yml

networks:
  whirrlyric:
    name: whirrlyric
    external: true
services:
  redis:
    image: redis:latest
    container_name: redis
    ports:
      - 6379:6379
    volumes:
      - ./data:/data
      - ./conf/redis.conf:/usr/local/conf/redis.conf
    networks:
      - whirrlyric
    restart: always
    command: redis-server /usr/local/conf/redis.conf
    environment:
      TZ: Asia/Seoul

redis.conf( -> redis 공식문서에서 제공되는 파일을 가져온 후 필요한 설정만 바꾸거나 추가한다

# 디폴트값 -> 루프백만 열기
#bind 127.0.0.1 -::1

# 개발용 설정 -> 다열기
bind 0.0.0.0

# 비밀번호 설정 -> 요청시마다 입력해야 함, spring boot에 설정 추가
requirepass ********************************
```

- NginX 설정

nginx.conf

```
user  nginx;
worker_processes  auto;
error_log  /var/log/nginx/error.log debug;
pid        /var/run/nginx.pid;
events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for" $scheme ';

    access_log  /var/log/nginx/access.log  main;

    #  기존 싸피경로, AWS IP로 직접 접근 모두 차단함 휘리릭으로 리다이렉트 http 전부차단
    server {
        listen 80;
        server_name ***.***.***.***; # <- AWS IP 적을 것
        location / {
            return 301 https://whirrlyric.n-e.kr;
        }

    }
    server {
        listen 443 ssl;
        server_name ***.***.***.***; # <- AWS IP 적을 것

        ssl_certificate /etc/nginx/ssl/certificate.pem; # 어떤 인증서를 쓸건지 작
        ssl_certificate_key /etc/nginx/ssl/private.key;
        ssl_trusted_certificate /etc/nginx/ssl/ca_bundle.crt;

        location / {
            return 301 https://whirrlyric.n-e.kr;
        }

    }

    server {
        listen 80;
        server_name whirrlyric.n-e.kr;
```

```
        location / {
            return 301 https://$server_name$request_uri;
        }
    }

    server {
        listen 443 ssl;
        server_name whirrlyric.n-e.kr;

        ssl_certificate /etc/nginx/ssl/certificate.pem; # 어떤 인증서 쓸건지 작성
        ssl_certificate_key /etc/nginx/ssl/private.key;
        ssl_trusted_certificate /etc/nginx/ssl/ca_bundle.crt;

        location / {
            proxy_pass http://fe:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;

        }
        location /api {
            proxy_pass http://be:8080;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        }

        location /jenkins {
            charset utf-8;
            proxy_no_cache 1;

            proxy_pass http://jenkins:8080;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
#            proxy_cache_bypass $http_upgrade;
        }
    }
    sendfile        on;
    keepalive_timeout  65;
    include /etc/nginx/conf.d/*.conf;
}
```

```
docker-compose.yml

networks:
  whirrlyric:
    name: whirrlyric
    external: true

services:
  nginx:
    container_name: nginx
    image: nginx
    restart: always
    ports:
      - 80:80
      - 443:443
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl:ro
    networks:
      - whirrlyric
    environment:
      - TZ=Asia/Seoul
```

Jenkins 컨테이너, 대시보드 설정

- 컨테이너 삭제가 자주 될 것으로 예상되면, DockerFIle을 작성하거나 초기 세팅이 모두 끝난 후 이미지를 commit해서 잘 보관한다. 컨테이너 삭제 시 볼륨 외의 작업이 모두 초기화된다.

1) Jenkins 컨테이너 내부로 접근하여 필요한 패키지 모두 설치

```
apt-get update
apt-get install vim curl tree ...

위에 설명된 도커 설치를 참조하여 jenkins 내부에도 도커 설치
```

2) Jenkins 대시보드 접속

```
nginx설정이 잘 되었다면 jenkins 환경변수 prefix로 설정한 주소로 접속이 가능하다.
ex) https://example/jenkins

* 접속이 안 될경우
- nginx 포트 매핑이 잘 되었는지 nginx.conf
- 프록시 설정 잘 되었는지 nginx.conf
- 환경변수 설정이 잘 되었는지 jenkins docker-compose.yml
- 도커 컨테이너가 하나의 네트워크로 묶여있는지 체크 docker network inspect *networkname
```

```
대시보드가 나오면 초기 비밀번호를 입력하고 필요한 플러그인 모두 설치
왼쪽 상단 new item -> 파이프라인 만들기 (FE, BE)
```

3) Jenkins 스크립트 작성

BE

```
pipeline {
    agent any

    stages {
        stage('clone') {
            steps {
                echo 'Branch : backend'
                echo 'Clone repository'
                git branch: 'backend', url: '********************', credential
            }
        }

        stage('docker build') {
            steps {
                echo 'docker build start'
                // 실행 중인 컨테이너가 있는지 확인 후 삭제
                // 스프링 컨테이너는 be 라는 이름으로 돌리고 있다
                sh 'docker ps -a | grep be && docker rm -f be || true'

                echo 'killed prev container'

                withCredentials([file(credentialsId: 'SPRING_SECRET', variabl
                    // 'SECRET_KEY' 환경 변수를 사용하여 필요한 작업 수행 - secret 파
                    sh 'cp $SECRET_KEY Backend/jayul/src/main/resources/'
                }

                withCredentials([file(credentialsId: 'SPRING_DATA_SOURCE', va
                    // 'SECRET_KEY' 환경 변수를 사용하여 필요한 작업 수행 - mysql 설정
                    sh 'cp $DB_KEY Backend/jayul/src/main/resources/'
                }

                // 백엔드 도커파일 위치 설정 -> 도커 이미지로 빌드 -> 도커 컨테이너 안쪽에
                // 도커파일에서 빌드, 실행까지
                // 도커 이미지 빌드 (태그,[옵션],컨텍스트)
                script {
                    docker.build('be', '-f Backend/jayul/Dockerfile Backend/j
                }
                sh 'docker run -e TZ=Asia/Seoul -d -p 8080:8080 --name be --n
            }
        }
```

```
    }
}
```

FE

```
pipeline {
    agent any

    stages {
        stage('clone') {
            steps {
                // 깃랩 frontend 가져오기
                echo 'git clone'
                git branch: 'frontend', credentialsId: 'GITLAB_CREDENTIAL', u
            }
        }
        stage('docker build') {
            steps {
                echo 'build'

                // 실행 중인 컨테이너가 있는지 확인 후 삭제
                sh 'docker ps -a | grep fe && docker rm -f fe || true'

                // 프론트 도커파일 위치 설정 -> 도커 이미지로 빌드 -> 도커 컨테이너 안쪽
                // 컨테이너 이름 fe

                withCredentials([file(credentialsId: '.env', variable: 'VITE_
                    // .env 복사 -> 클론해온폴더 아래 FE폴더로
                    sh 'cp $VITE_ENV FrontEnd/fe-project/'
                }

                // 도커 이미지 빌드
                script {
                    docker.build('fe', '-f FrontEnd/fe-project/Dockerfile Fro
                }

                // 도커 컨테이너 실행
                sh 'docker run -e TZ=Asia/Seoul -d -p 3000:3000 --name fe --n
            }
        }
        stage('deploy') {
            steps {
                echo 'deploy'
            }
        }
```

```
    }
}
```

4) Credential 추가, gitlab webhook 설정, 기본 경로 변경

```
대시보드 -> 젠킨스 관리 -> credential

- FE 환경변수 파일
- BE 스프링부트 설정파일
- gitlab 연동용 credential => gitlab에서 프로젝트 token발급해서 이곳에 추가
그 외 필요한 credential을 모두 추가한다.

대시보드 -> 젠킨스 관리 -> 시스템

- Jenkins Location
-> 등록한 도메인으로 입력한다 ex) https://example.com/jenkins
- 이제 젠킨스 웹서버는 모든 정적파일을 이 경로로 제공한다.
```

5) gitlab webhook 연동

```
gitlab 프로젝트 접속 -> setting -> webhooks

- Add new webhook
- 파이프라인 작성 시 생성한 비밀번호 추가, url 추가
- 파이프라인마다 원하는 대로 옵션을 주고 여러개 만들기
```

DockerFile 작성

BE

```
FROM gradle:8.7-jdk17 as builder
WORKDIR /app
COPY build.gradle settings.gradle ./
RUN gradle build -x test --parallel --continue > /dev/null 2>&1 || true # 도커

COPY . ./
RUN gradle build -x test --parallel

FROM azul/zulu-openjdk:17
WORKDIR /app
COPY --from=builder /app/build/libs/jayul-0.0.1-SNAPSHOT.jar .
```

```
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=prod", "jayul-0.0.1-SNA
```

FE

```
FROM node:20 as builder
WORKDIR /app
COPY package*.json ./
COPY yarn.lock ./
RUN yarn
COPY . ./
RUN yarn build

FROM nginx:latest
RUN mkdir /app
WORKDIR /app
COPY  --from=builder /app/dist /app/dist
RUN rm /etc/nginx/conf.d/default.conf
COPY ./default.conf /etc/nginx/conf.d
EXPOSE 3000

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

외부서비스

Suno-API : https://github.com/SunoAI-API/Suno-API

> https://github.com/SunoAI-API/Suno-API

- 스프링 서버에서 호출