**[SSAFY 12기 A209팀]**

1. **시스템 환경 설정(사용한 JVM, 웹서버, WAS 제품 등의 종류와 설정 값, 버전(IDE버전 포함) 기재)**
   1. **Backend**
      1. JDK
         1. Oracle OpenJDK 21.0.5
      2. WAS(Web Application Server)
         1. Apache Tomcat (Spring Boot 내장)
         2. port : 8080 (Blocked)
         3. context path : /
         4. encoding : UTF-8
      3. Development Tool
         1. IDE : InteliJ IDEA 2024.3.3 (Ultimate Edtion)
         2. Build Tool : Gradle 8.11
      4. Framework
         1. Spring Boot : 3.4.1
      5. Database
         1. MariaDB
         2. Elasticsearch 8.17.2
            1. nori-tokenizer plugin
         3. H2 (for Test)
      6. Dependencies
         1. Spring Security
         2. Spring Data JPA
         3. Spring Data Elasticsearch
         4. Spring Cloud AWS S3 3.0.0
         5. Firebase Admin 9.4.2
         6. JUnit 5.10.0
         7. Lombok
      7. Util
         1. Checkstyle 8.42 (네이버 규칙)
   2. **Frontend**
      1. Node.js 22.13.0
      2. Server
         1. Vite preview
         2. Port
            1. 사용자 페이지 : 80
            2. 관리자 페이지 : 8100
      3. Development Tool
         1. Vite
         2. ESLint
         3. Prettier
         4. Git
         5. Visual Studio Code
      4. Dependencies
         1. Vue.js
         2. Vue Router
         3. Pinia
         4. Axios
         5. Firebase
         6. Tailwind CSS
         7. Vite svg loader
         8. postcss
   3. **Infra & CI/CD** 
      1. Linux Ubuntu
         1. Docker
      2. Raspberry Pi 5
         1. Docker
         2. Jenkins
            1. Oracle OpenJDK 21.0.5 for build
            2. NodeJS 22.13.0 for build
   4. **Crawling System** 
      1. AWS EventBridge
      2. AWS Lambda
      3. AWS ECR
      4. Python 3.9
         1. Development tool
            1. PyCharm 24.3.1.1
         2. Library
            1. selenium
            2. openai
            3. pymysql
            4. beautifulsoup4
2. **빌드 시 사용되는 환경 변수 등의 내용 상세 기재**
   1. **Backend** 
      1. .env
         1. 경로
            1. 프로젝트 루트 디렉토리 (for test)
            2. Jenkins 마운트 볼륨 (for build)
         2. IDE 테스트 시 run configuration에서 해당 파일 지정
         3. 항목
            1. DB\_URL
            2. DB\_USERNAME
            3. DB\_PASSWORD
            4. AWS\_ACCESS\_KEY
            5. AWS\_SECRET\_KEY
            6. AWS\_S3\_BUCKET\_NAME
            7. AWS\_REGION=ap-northeast-2
            8. S3\_FILE\_SALT
            9. ELASTIC\_PASSWORD
            10. ELASTICSEARCH\_HOST
            11. PROFILE

(개발 환경 : test, 배포 환경 : prod)

* + 1. firebase-adminsdk.json  
       
  1. **Frontend** 
     1. front.env
        1. 경로
           1. S12P11A209/devoot-backend/ (for test)
           2. S12P11A209/devoot-frontend/ (for test)
           3. Jenkins 마운트 볼륨 (for build)
        2. 항목
           1. VITE\_API\_KEY

filrebase authentication 키

* + - * 1. VITE\_AUTH\_DOMAIN
        2. VITE\_PROJECT\_ID
        3. VITE\_STORAGE\_BUCKET
        4. VITE\_MESSAGING\_SENDER\_ID
        5. VITE\_APP\_ID

1. **배포 시 특이사항 기재**
   1. 라즈베리파이
      1. Docker 설치
      2. docker-compose.yml
      3. Jenkins
         1. 볼륨 마운트
            1. /home/jenkins → /var/jenkins\_home
            2. /var/run/docker.sock → /var/run/docker.sock

Jenkins 내에서 호스트의 도커 데몬 조작

* + - * 1. /usr/bin/docker → /usr/bin/docker

Jenkins 컨테이너 내에서 호스트의 Docker CLI 사용

* + - * 1. /home/env → /env

환경 변수 파일, SSL 인증서 참조

* + - * 1. /home/jdk-21.0.5 → /opt/jdk-21

JDK 빌드를 위한 라이브러리 공유

* + - 1. JDK Configuration
         1. Java Home 경로 설정 → /opt/jdk-21
      2. Credentials
         1. git\_account ( type : usernamepassword )
         2. docker\_account ( type : usernamepassword )
      3. Backend Pipeline  
         
      4. 절차
         1. 깃랩 리포지토리 클론
         2. 환경 변수, SSL 인증서 파일 워크스페이스로 복사
         3. spring boot 프로젝트 gradle 빌드
         4. Docker image 빌드 및 push

라즈베리파이(aarch64)에서 빌드하고 EC2(amd64)에서 실행되므로 tonistiigi/binfmt를 통해 qemu 환경의 컨테이너를 실행하고, 지원되는 실행 환경 설치 후 buildx를 통해 amd64 아키텍처에서 실행 가능한 dockerimage 빌드

* + - * 1. AWS EC2에서 실행 중인 docker로 context 생성 및 switch ( SSL 인증 )
        2. 실행 중이던 기존 컨테이너 및 이미지 삭제
        3. compose up -d를 통해 애플리케이션 pull 및 실행
        4. elasticsearch 컨테이너에 nori-tokenizer 플러그인 설치
      1. frontend pipeline  
         
      2. 절차
         1. 깃랩 리포지토리 클론
         2. @rollup/rollup-linux-64-gnu 설치
         3. npm build
         4. AWS EC2에서 실행 중인 docker로 context 생성 및 switch ( SSL 인증 )
         5. 실행 중인 docker container, image 정리
         6. 환경 변수 파일 복사
         7. 사용자 페이지 프로젝트 dockerimage 빌드 및 push
         8. 관리자 페이지 프로젝트 dockerimage 빌드 및 push
         9. docker compose로 배포
         10. elasticsearch 컨테이너에 nori-tokenizer 플러그인 설치
      3. EC2
         1. 포트 개방 (ufw)

80(사용자 웹 페이지), 2376(원격 docker daemon context TLS 접속), 8100(관리자 웹 페이지)

* + - * 1. Docker

원격 제어를 위한 TLS/SSL 인증서 설치

ca.pem, server-cert.pem, server-key.pem 파일을 /etc/docker/ssl/로 이동

server-key.pem 파일에 권한 부여 → chmod 600

docker 재시작

unable to configure the Docker daemon 에러 발생 시 daemon.json 파일에서 hosts 설정 삭제

NodeJS 22.13.0 ( for build frontend )

볼륨 마운트 ( ec2에서 docker volume create로 볼륨 사전 생성 )

elastic\_config → /usr/share/elasticsearch/config

esdata → /usr/share/elasticsearch/data

cert → /usr/share/elasticsearch/config/certs

1. **DB 접속 정보 등 프로젝트(ERD)에 활용되는 주요 계정 및 프로퍼티가 정의된 파일 목록**
   1. .env
   2. front.env
   3. firebase-adminsdk.json