

포팅 메뉴얼

▼ 목차

1. 사용한 버전

2. Infra

[SonarQube](#)

[Database](#)

[Monitoring](#)

[Filebrowser](#)

[jenkins](#)

[Test-DATABASE](#)

[Backend](#)

[Frontend](#)

[Data](#)

[Product](#)

1. 사용한 버전

분류	환경	버전
Backend	Spring Boot	2.7.10
	JPA	5.0.0
	JDK	openjdk - 11.0.18
	Spring Security	5.7.6
	OAuth2	2.7.9
	QueryDSL	5.0.0
	actuator	2.7.10
FrontEnd	React.js	18.2.0
	Next.js	13.3.0
	Redux-toolkit	1.9.4
	node.js	18.16.0
	styled-components	5.3.9
	typescript	5.0.4
Data	python	3.9
	FastAPI	0.95.2
DataBase	MySQL	8.0.28
	Redis	7.0.8
Infra	Docker	23.0.0
	jenkin	2.375.2
	ubuntu	20.04 LTS
	sonarqube scanner	3.5.0.2730
	filebrowser	latest
	grafana	latest
	prometheus	latest

2. Infra

▼ SonarQube

build.gradle에 있는 test 설정을 통해 Jacoco Coverage를 측정하여 xml 파일을 생성하고 SonarQube 분석을 통해 xml 파일 전송하여 측정한 Coverage 전송하고 개발자에게 MatterMost 알림 전송

▼ Docker-compose - SonarQube

```
sonarqube:
  image: sonarqube:lts
  container_name: sonarqube
  ports:
    - "9000:9000"
  ulimits:
    nofile:
      soft: 262144
      hard: 262144
  networks:
    - sonar_network
  environment:
    - sonar.jdbc.url=jdbc:postgresql://sonar-db:5432/sonar
  volumes:
    - sonarqube_conf:/opt/sonarqube/conf
    - sonarqube_data:/opt/sonarqube/data
    - sonarqube_extensions:/opt/sonarqube/extensions
    - sonarqube_logs:/opt/sonarqube/logs

sonar-db:
  image: postgres
  container_name: postgres_sonar
  ports:
    - "5432:5432"
  networks:
    - sonar_network
  environment:
    - POSTGRES_USER=sonar
    - POSTGRES_PASSWORD=sonar
  volumes:
    - postgresql:/var/lib/postgresql
    - postgresql_data:/var/lib/postgresql/data

networks:
  sonar_network
volumes:
  sonarqube_conf:
  sonarqube_data:
  sonarqube_extensions:
  sonarqube_logs:
  postgresql:
  postgresql_data:
```

▼ Jenkins Pipeline

```
pipeline {
  agent any
  environment {
    GIT_URL = "[Git 주소]"
    BACKEND_PROJECT_PATH = "${WORKSPACE}/backend"
  }
  stages{
    stage('Pull') {
      steps {
        script {
          git url: "${GIT_URL}", branch: "develop/be", credentialsId: 'gitlab-user', poll: true, changelog: true
        }
      }
    }
    stage("Copy Setting File"){
      steps{
        script{
          sh "pwd"
          sh "ls"
          sh "cp -r -f resources ${BACKEND_PROJECT_PATH}/src/main"
          sh "rm -rf ${BACKEND_PROJECT_PATH}/src/test/resources"
          sh "cp -r -f testResources ${BACKEND_PROJECT_PATH}/src/test/resources"
          sh "cp -f gradleResources/gradle.properties ${BACKEND_PROJECT_PATH}"
        }
      }
    }
    stage('Jacoco Coverage Test') {
```

```

        steps {
            script{
                try{
                    dir("${BACKEND_PROJECT_PATH}") {
                        sh "chmod +x gradlew"
                        sh './gradlew clean test'
                    }
                } catch(Exception e){
                    echo "Test 실패가 발견되었습니다."
                }
            }
        }
    }

    stage('SonarQube Analysis') {
        steps {
            dir("${BACKEND_PROJECT_PATH}") {
                sh "chmod +x gradlew"
                sh './gradlew sonar'
            }
        }
    }

    stage('MatterMost Notification'){
        steps {
            script {
                def decodedJobName = java.net.URLDecoder.decode(env.JOB_NAME, "UTF-8")
                def messageText = "### MergeRequest Event\n" +
                    "${decodedJobName} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)\n" +
                    "\n" +
                    "#### <[소나큐브 주소]|소나큐브 분석 바로가기>"

                def payload = [
                    username: 'Jenkins',
                    icon_url: 'https://jenkins.io/images/logos/jenkins/jenkins.png',
                    text: messageText
                ]

                def response = httpRequest(
                    url: '[웹hookurl]',
                    httpMode: 'POST',
                    contentType: 'APPLICATION_JSON',
                    requestBody: groovy.json.JsonOutput.toJson(payload),
                    validResponseCodes: '200:299'
                )
                echo "Response status: ${response.status}"
                echo "Response content: ${response.content}"
            }
        }
    }
}
}

```

▼ Jacoco build.gradle 설정

```

plugins {
    id 'jacoco'
}
jacoco {
    toolVersion = '0.8.5'
}
test {
    jacoco {
        // 아래 설정들은 모두 기본 값. 따라서 변경할 것이 없다면 적어주지 않아도 됨
        enabled = true
        destinationFile = file("${buildDir}/jacoco/${name}.exec")
        includes = []
        excludes = []
        excludeClassLoaders = []
        includeNoLocationClasses = false
        sessionId = "<auto-generated value>"
    }
}

```

```

        dumpOnExit = true
        classDumpDir = null
        output = JacocoTaskExtension.Output.FILE
        address = "localhost"
        port = 6300
        jmx = false
    }
    finalizedBy 'jacocoTestReport'
}
jacocoTestReport {
    reports {
        html.enabled true // html 만들어 - 로컬에서 쉽게 보기 위함
        xml.enabled true // xml 만들어 - 소나큐브 연동 위함
        csv.enabled false // csv 안 만들어

        // xml.destination file("${buildDir}/jacoco/result.xml") // 여기 저장할 것이라는 뜻.
    }
    finalizedBy 'jacocoTestCoverageVerification'
}
jacocoTestCoverageVerification {
    violationRules {
        rule {
            enabled = true // 이 rule을 적용할 것이다.
            element = 'CLASS' // class 단위로

            // 브랜치 커버리지 최소 50%
            limit {
                counter = 'BRANCH'
                value = 'COVEREDRATIO'
                minimum = 50
            }

            // 라인 커버리지 최소한 80%
            limit {
                counter = 'LINE'
                value = 'COVEREDRATIO'
                minimum = 80
            }

            // 빈 줄을 제외한 코드의 라인수 최대 300라인
            limit {
                counter = 'LINE'
                value = 'TOTALCOUNT'
                maximum = 300
            }

            // 커버리지 체크를 제외할 클래스들
            excludes = [
                '**/oauth/**',
                '**/auth/**',
                '**/entity/*.java',
                '*.CustomOAuth2SuccessHandler.*'
            ]
        }
    }
}
}

```

▼ 디렉토리 구조

```

Sonar-PipeLine
├── README.md
├── backend
│   ├── Dockerfile
│   ├── Jenkinsfile
│   ├── build
│   ├── build.gradle
│   ├── docker-compose.yml
│   ├── gradle
│   ├── gradle.properties
│   ├── gradlew
│   ├── gradlew.bat
│   ├── settings.gradle
│   └── src
├── gradleResources
│   └── gradle.properties
├── resources
│   ├── application-deploy.yml
│   └── application.yml

```

```
└─ testResources
   └─ application.yml
```

▼ 소나 properties

▼ gradle.properties

```
systemProp.sonar.projectKey=[프로젝트키]
systemProp.sonar.host.url=[소나큐브 주소]
systemProp.sonar.login=[소나큐브 로그인 키]
systemProp.sonar.exclusions = **/auth/**/*.java
```

▼ sonar.properties

```
# must be unique in a given SonarQube instance
sonar.projectKey=[프로젝트키]
# this is the name and version displayed in the SonarQube UI. Was mandatory prior to SonarQube 6.1.
sonar.projectName=[프로젝트 이름]
sonar.projectVersion=1.0
sonar.host.url=[소나서버 url]
sonar.login=[로그인 토큰]

# Path is relative to the sonar-project.properties file. Replace "\" by "/" on Windows.
# This property is optional if sonar.modules is set.
sonar.sources=.

# Encoding of the source code. Default is default system encoding
sonar.sourceEncoding=UTF-8
```

▼ Database

※ MySQL은 AWS RDS로 진행함.

1. AWS RDS에서 MySQL을 선택.

데이터베이스 생성

데이터베이스 생성 방식 선택 정보

☒ 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 높입니다.

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 이름, 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 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 끝낼 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.

▼ Monitoring

docker-compose

```
version: "3"
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus_service
    # command:
    #   - '--config.file=/etc/prometheus/prometheus.yml'
    #   - '--web.route-prefix=/collection/'
```

```
# - '--web.external-url=/collection/'
volumes:
  - ./prometheus:/etc/prometheus
ports:
  - 9998:9090
networks:
  - mysql_application_network
restart: always
grafana:
  user: root
  image: grafana/grafana
  container_name: grafana_service
  environment:
    - GF_SERVER_ROOT_URL=[접속할 url]
    - GF_SECURITY_ADMIN_USER=admin
    - GF_SECURITY_ADMIN_PASSWORD=ssafy1234
  volumes:
    - ./grafana:/var/lib/grafana
  ports:
    - 3001:3000
  networks:
    - mysql_application_network
  restart: always
mysqld-exporter:
  image: prom/mysqld-exporter
  command:
    - --collect.info_schema.processlist
    - --collect.info_schema.query_response_time
  container_name: mysqld-exporter
  environment:
    - DATA_SOURCE_NAME=[db 주소]
  ports:
    - 9104:9104

networks:
  mysql_application_network:
    external: true
```

prometheus.yml

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: chpo
    metrics_path: '[api 주소]'
    scrape_interval: 10s
    static_configs:
      - targets: ['[주소]']
  - job_name: rds
    metrics_path: '/metrics'
    scrape_interval: 10s
    static_configs:
      - targets: ['[주소]']
```

▼ Filebrowser

docker-compose

```
## docker-compose
version: "3"
services:
  filebrowser:
    image: filebrowser/filebrowser
    container_name: filebrowser
    restart: unless-stopped
    #environment:
    #- FB_BASEURL=fb
    volumes:
```



```

- ./filebrowser.json:/filebrowser.json
- ./database.db:/database.db
- /home/ubuntu:/srv
ports:
- 9999:80

```

filebrowser.json

```

{
  "port": 80,
  "baseUrl": "/fb",
  "database": "/database.db",
  "scope": "/srv",
  "root": "/srv",
  "allowCommands": true,
  "allowEdit": true,
  "allowNew": true,
  "commands": []
}

```

▼ jenkins

dockerfile

```

#Dockerfile
FROM jenkins/jenkins:ls
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 docker-compose-plugin
RUN usermod -s /bin/bash jenkins

USER jenkins

```

▼ Test-DATABASE

docker-compose

```

version: '3'
services:
  mysql:
    image: mysql:8.0
    container_name: test-mysql
    ports:
      - 13306:3306 # HOST:CONTAINER
    environment:
      MYSQL_ROOT_HOST: "%"
      MYSQL_ROOT_PASSWORD: [비밀번호]
      MYSQL_DATABASE: "free"
      TZ: Asia/Seoul
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    volumes:
      - ./data:/var/lib/mysql
      - ./mysql-init.d:/docker-entrypoint-initdb.d
    networks:
      - mysql_application_network
networks:
  mysql_application_network:
    external: true

```

▼ Backend

▼ docker-compose - backend

```

version: "3"
services:
  app:
    image: sh80165/free-springboot
    # build:
    #   context: .
    #   dockerfile: ./Dockerfile
    ports:
      - 8080:8080
    networks:
      - mysql_application_network
    restart: always

networks:
  mysql_application_network:
    external: true
    # name: mysql_application_network

```

▼ JenkinsFile - backend

```

pipeline {
  agent any

  // tools{
  //   gradle "spring boot gradle"
  // }

  stages {
    stage("Set Variable") {
      steps {
        script {
          IMAGE_NAME = "sh80165/free-springboot"
          IMAGE_STORAGE = "https://registry.hub.docker.com"
          IMAGE_STORAGE_CREDENTIAL = "docker-hub"
          // APPLICATION_YML_PATH = "/home/ubuntu/jenkins/workspace/backend-pipeline"
          CONTAINER_NAME = "backend-app-1"
          PROJECT_DIR = "backend/"
          DOCKER_FILE_PATH = "./backend"
        }
      }
    }

    stage("Copy Setting File"){
      steps{
        script{
          sh "pwd"
          sh "ls"
          sh "cp -r -f resources ${PROJECT_DIR}"
          sh "cp -f gradleResources/gradle.properties ${PROJECT_DIR}"
        }
        // dir("${APPLICATION_YML_PATH}") {
        //   sh "pwd"
        //   sh "ls"
        //   sh "cp -r -f resources ${PROJECT_DIR}/resources"
        // }
      }
    }

    // //스프링 빌드
    // stage("Clean Build Test") {
    //   steps {
    //     dir("${PROJECT_DIR}") {
    //       sh "pwd"
    //       sh "chmod +x gradlew"
    //       sh "./gradlew clean build -x test"
    //       sh "ls -al ./build"
    //     }
    //   }
    // }

    // 소나큐브 빌드
    stage('SonarQube Analysis') {
      steps {
        dir("${PROJECT_DIR}") {
          sh "chmod +x gradlew"
          sh './gradlew sonar'
        }
      }
    }
  }
}

```

```

//    }
// }

//기존 컨테이너 내리고 이미지 삭제
stage("container down&image remove"){
    // 컴포즈 down 후 컨테이너 삭제.
    steps{
        dir("${DOCKER_FILE_PATH}"){

            // sh "docker compose ps -q | xargs -r docker compose down"

            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"

            sh "docker images ${IMAGE_NAME} -q | xargs -r docker rmi -f"

        }
    }
}

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

//
//    stage("Push Container Image") {
//        steps {
//            script {
//
//                docker.withRegistry("", "${IMAGE_STORAGE_CREDENTIAL}") {
//                    image.push("latest")
//                }
//            }
//        }
//    }
// //
stage("Server Run") {
    steps {
        dir("${DOCKER_FILE_PATH}"){
            sh "docker compose up -d --build"
        }
    }
}
}
}

```

▼ DockerFile - backend

```

FROM openjdk:11-jdk

# FROM gradle:7.6.1-jdk11

ENV APP_HOME=/usr/app

WORKDIR $APP_HOME

COPY . .

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

RUN chmod +x gradlew

RUN ./gradlew clean build -x test

COPY ./resources ./resources

```

```

RUN cp ./build/libs/*.jar application.jar

EXPOSE 8080

ENTRYPOINT ["java", "-Duser.timezone=Asia/Seoul", "-jar", "-Dspring.config.location=resources/application.yml,resources/application-de

```

▼ 디렉토리 구조

```

Backend-pipeline
├── README.md
├── backend
│   ├── Dockerfile
│   ├── Jenkinsfile
│   ├── build
│   ├── build.gradle
│   ├── docker-compose.yml
│   ├── gradle
│   ├── gradle.properties
│   ├── gradlew
│   ├── gradlew.bat
│   ├── resources
│   ├── settings.gradle
│   └── src
├── backend@tmp
├── gradleResources
│   └── gradle.properties
├── resources
│   ├── application-deploy.yml
│   └── application.yml

```

▼ Test환경

```

spring:
  h2:
    console:
      enabled: true
      path: /h2-console
  jpa:
    hibernate:
      ddl_auto: create
    database: h2
    generate-ddl: off
    defer-datasource-initialization: true
  datasource:
    driver-class-name: org.h2.Driver
    url: jdbc:h2:mem:testdb;MODE=MySQL;
    username: SA
    password:

```

▼ Frontend

▼ docker-compose - frontend

```

version: "3"
services:
  app:
    image: sh80165/free-react
    # build:
    #   context: .
    #   dockerfile: ./Dockerfile
    ports:
      - 3000:3000
    # networks:
    #   - mysql_application_network
    restart: always

# networks:
#   mysql_application_network:
#     external: true
#     # name: mysql_application_network

```

▼ JenkinsFile - frontend

```
pipeline {
    agent any

    stages {
        stage("Set Variable") {
            steps {
                script {
                    IMAGE_NAME = "sh80165/free-react"
                    IMAGE_STORAGE = "https://registry.hub.docker.com"
                    IMAGE_STORAGE_CREDENTIAL = "docker-hub"
                    NODE_BUILD_PATH = "./build"
                    APPLICATION_ENV_PATH = "/var/jenkins_home/workspace"
                    // ESLINTIGNORE_PATH = "/var/jenkins_home/workspace/free-frontend"
                    CONTAINER_NAME = "frontend-app-1"
                    PROJECT_DIR = "frontend"
                    DOCKER_FILE_PATH = "./frontend"
                }
            }
        }

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

        stage("env copy") {
            steps {
                script {
                    sh "pwd"
                    sh "cp -f .env frontend/.env"
                }
            }
        }

        stage("Clean Container&Image") {
            steps {
                dir("${DOCKER_FILE_PATH}") {

                    // sh "docker compose ps -q | xargs -r docker compose down"
                    // // 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 {
                dir("${DOCKER_FILE_PATH}"){
                    sh "docker compose up -d"
                }
            }
        }
    }
}
}

```

▼ DockerFile - frontend

```

# 노드 버전 지정
FROM node:18-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" ]

```

▼ 디렉토리 구조

```

Frontend-pipeline
├─ README.md
├─ frontend
│   ├── Dockerfile
│   ├── Jenkinsfile
│   ├── README.md
│   ├── components
│   ├── docker-compose.yml
│   ├── next.config.js
│   ├── package-lock.json
│   ├── package.json
│   ├── pages
│   ├── public
│   ├── redux
│   ├── styles
│   ├── tsconfig.json
│   └─ utils
├─ frontend@tmp
└─ package-lock.json

```

▼ .env 설정 파일

```

NEXT_PUBLIC_API_URL="[백엔드 api 주소]"

NEXT_PUBLIC_MODE='dev'

```

▼ Data

▼ docker-compose - data``

```

version: "3.9"

services:
  fastapi_app:
    build:
      dockerfile: Dockerfile
    image: sh80165/free-data
    ports:
      - "8000:8000"
    container_name: my_fastapi_container
    networks:
      - mysql_application_network
networks:
  mysql_application_network:
    external: true

```

▼ JenkinsFile - data

```

pipeline {
  agent any

  stages {
    stage("Set Variable") {
      steps {
        script {
          IMAGE_NAME = "sh80165/free-data"
          IMAGE_STORAGE = "https://registry.hub.docker.com"
          IMAGE_STORAGE_CREDENTIAL = "docker-hub"
          // ESLINTIGNORE_PATH = "/var/jenkins_home/workspace/free-frontend"
          CONTAINER_NAME = "my_fastapi_container"
          PROJECT_DIR = "data"
          DOCKER_FILE_PATH = "./data"
        }
      }
    }

    stage("Copy Setting File") {
      steps {
        script {
          sh "pwd"
          sh "cp -f env.json ${PROJECT_DIR}"
        }
      }
    }

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

    // 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 {
        dir("${DOCKER_FILE_PATH}") {

          // sh "docker compose ps -q | xargs -r docker compose down"
          // // //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"
        sh "cat main.py"
        sh "docker build --no-cache -t ${IMAGE_NAME} ."
      }
    }
  }
}

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

stage("Server Run") {
  steps {
    dir("${DOCKER_FILE_PATH}") {
      sh "docker compose up -d"
    }
  }
}
}
}

```

▼ DockerFile - data

```

# 기본 이미지 선택 (Python 3.9 버전 사용)
FROM python:3.9

# 작업 디렉터리 설정
WORKDIR /usr/bin

# 의존성 설치
RUN apt-get -y update
RUN apt install wget
RUN apt install unzip
RUN wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
RUN apt -y install ./google-chrome-stable_current_amd64.deb
RUN wget -O /tmp/chromedriver.zip http://chromedriver.storage.googleapis.com/` curl -sS chromedriver.storage.googleapis.com/LATEST_
RUN mkdir chrome
RUN unzip /tmp/chromedriver.zip chromedriver -d /usr/bin/chrome

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 소스 코드 복사
COPY . .
RUN cat main.py

# 애플리케이션 실행
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

```

▼ 디렉토리 구조

```

.
├── README.md
├── data
│   ├── Dockerfile
│   ├── Jenkinsfile
│   └── docker-compose.yml

```



```

|   |— env.json
|   |— main.py
|   |— requirements.txt
|— data@tmp
|— env.json

```

▼ env.json 설정 파일

```

{
  "token" : "token [토큰]"
}

```

▼ Product

▼ docker-compose - main

```

version: "3"
services:
  backend:
    image: sh80165/product-free-springboot
    container_name: product-be
    restart: always
    ports:
      - 8080:8080
    networks:
      - app-network
  frontend:
    image: sh80165/product-free-react
    container_name: product-fe
    restart: always
    ports:
      - 3000:3000
    networks:
      - app-network
  data:
    image: sh80165/product-free-data
    container_name: product-data
    restart: always
    ports:
      - 8000:8000
    networks:
      - app-network
networks:
  app-network:
    external: true

```

▼ JenkinsFile - main

```

pipeline {
  agent any

  stages {
    stage("Set Variable") {
      steps {
        script {
          IMAGE_NAME_FE = "sh80165/product-free-react"
          IMAGE_NAME_BE = "sh80165/product-free-springboot"
          IMAGE_NAME_DATA = "sh80165/product-free-data"
          IMAGE_STORAGE = "https://registry.hub.docker.com"
          IMAGE_STORAGE_CREDENTIAL = "docker-hub"
          SSH_CONNECTION = "${env.SSH_CONNECTION}"
          SSH_CONNECTION_CREDENTIAL = "product-server-ssh-credential"
          APPLICATION_YML_PATH = "/var/jenkins_home/workspace"
          CONTAINER_NAME_FE = "product_FE"
          CONTAINER_NAME_BE = "product_BE"
          CONTAINER_NAME_DATA = "product_data"
          PROJECT_DIR_FE = "frontend/"
          PROJECT_DIR_BE = "backend/"
          PROJECT_DIR_DATA = "data/"
        }
      }
    }
  }
}

```

```

}
//scp를 이용해서 docker compose, script 파일 전송.
stage("Send file") {
    steps {
        sshagent([SSH_CONNECTION_CREDENTIAL]) {

            sh "scp -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null docker-compose.yml container-start.sh ${SSH}
        }
    }
}

// stage("Send file") {
//     stage("docker compose yml"){
//         steps {
//             sshagent([SSH_CONNECTION_CREDENTIAL]) {

//                 sh "scp -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null docker-compose.yml container
//             }
//         }
//     }
// }

// stage("script"){
//     steps {
//         script {
//             sh "pwd"
//         }
//     }
// }

// }
//기존 이미지 삭제
stage("image rm"){
    steps{
        sh "docker images ${IMAGE_NAME_FE} -q | xargs -r docker rmi -f"
        sh "docker images ${IMAGE_NAME_BE} -q | xargs -r docker rmi -f"
        sh "docker images ${IMAGE_NAME_DATA} -q | xargs -r docker rmi -f"
    }
}

stage("BE image build&push"){
    steps{
        //설정파일 카피
        script{
            sh "cp -r -f resources ${PROJECT_DIR_BE}"
        }
        //도커 이미지 빌드
        dir("${PROJECT_DIR_BE}") {
            script{
                // image = docker.build("${IMAGE_NAME_BE}") {
                //     args "--no-cache"
                // }
                sh "docker build --no-cache -t ${IMAGE_NAME_BE} ."
            }
        }
        // withCredentials([string(credentialsId: "${IMAGE_STORAGE_CREDENTIAL}")]) {
        //도커 허브에 푸시
        script{
            docker.withRegistry("", "${IMAGE_STORAGE_CREDENTIAL}") {
                // image.push("latest")
                sh "docker push ${IMAGE_NAME_BE}"
            }
        }
    }
}

stage("FE image build&push"){
    steps{
        script{
            sh "cp -f .env ${PROJECT_DIR_FE}.env"
        }

        //도커 이미지 빌드
        dir("${PROJECT_DIR_FE}") {
            script {
                // image = docker.build("${IMAGE_NAME_FE}") {
                //     args "--no-cache"
                // }
                sh "docker build --no-cache -t ${IMAGE_NAME_FE} ."
            }
        }
    }
}

```



```

|   |   ├── Dockerfile
|   |   ├── Jenkinsfile
|   |   ├── docker-compose.yml
|   |   ├── env.json
|   |   ├── main.py
|   |   └── requirements.txt
|   ├── data@tmp
|   ├── docker-compose.yml
|   ├── docs
|   |   ├── 102조_중간발표.pdf
|   |   └── README.md
|   ├── env.json
|   ├── frontend
|   |   ├── Dockerfile
|   |   ├── Jenkinsfile
|   |   ├── README.md
|   |   ├── components
|   |   ├── docker-compose.yml
|   |   ├── next.config.js
|   |   ├── package-lock.json
|   |   ├── package.json
|   |   ├── pages
|   |   ├── public
|   |   ├── redux
|   |   ├── styles
|   |   ├── tsconfig.json
|   |   └── utils
|   ├── frontend@tmp
|   ├── package-lock.json
|   └── resources
|       ├── application-deploy.yml
|       └── application.yml

```

▼ container-start.sh

```

#!/bin/bash

# DOCKER_COMPOSE_FILE="your-docker-compose-file.yml"

BE_IMAGE_NAME="sh80165/product-free-springboot"
FE_IMAGE_NAME="sh80165/product-free-react"
DATA_IMAGE_NAME="sh80165/product-free-data"

# 컨테이너 실행 여부 확인
if [[ "$(sudo docker compose ps -q)" ]]; then
    echo "컨테이너가 이미 실행 중입니다. 컨테이너를 중지합니다."
    sudo docker compose down
fi

sleep 1

# 이미지 파일 삭제.
echo "기존 이미지 파일을 삭제 합니다."
sudo docker images ${BE_IMAGE_NAME} -q | xargs -r docker rmi -f
sudo docker images ${FE_IMAGE_NAME} -q | xargs -r docker rmi -f
sudo docker images ${DATA_IMAGE_NAME} -q | xargs -r docker rmi -f

sleep 1

# 이미지 pull
echo "이미지를 받아옵니다."
sudo docker pull ${BE_IMAGE_NAME}
sudo docker pull ${FE_IMAGE_NAME}
sudo docker pull ${DATA_IMAGE_NAME}

# 컨테이너 실행
echo "컨테이너를 실행합니다."
sudo docker compose up -d

```

▼ application-deploy.yml

```

spring:
  datasource:

```

```

url: "jdbc:mysql://database-1.c6nqdmwmr1mh.ap-northeast-2.rds.amazonaws.com:3306/free?serverTimezone=UTC&useLegacyDatetimeCode=
username: admin
password: ssafy80165!
driver-class-name: com.mysql.cj.jdbc.Driver
jpa:
  hibernate:
    ddl_auto: none
  properties:
    hibernate:
      globally_quoted_identifiers: true
      format_sql: false
      show_sql: false #sys.out으로 sql 로그 남겨줌.
      generate_statistics: true # 쿼리 수행 통계를 확인할 수 있다.
  jdbc:
    time_zone: Asia/Seoul
  database-platform: org.hibernate.dialect.MySQL8Dialect
  open-in-view: false
##시큐리티 설정
security:
  oauth2:
    client:
      registration:
        github:
          client-id: ffb3ee1b916a40075bd1
          client-secret: 811d2706e650f6b16d0b5d549e03e05367a98d2e
          redirect-uri: https://chpo.kr/api/login/oauth2/code/github
          scope:
            - read:user
            - user:email
##레디스 설정
redis:
  # lettuce:
  #   pool:
  #     max-active: 10
  #     max-idle: 10
  #     min-idle: 2
  port: 6379
  host: redis_service
  # password: 'abcde'

##fast api 관련
external-service:
  url: http://product-data:8000

##JWT관련
auth:
  tokenSecret: free-secret-key
  redirectPage: https://chpo.kr/redirect
  redirectPageServer: https://chpo.kr/redirect

#MatterMost 관련
notification:
  mattermost:
    enabled: true # mmSender를 사용할 지 여부, false면 알림이 오지 않는다
    webhook-url: https://meeting.ssafy.com/hooks/su6es41rytd13mdg7dz8g91bme # 위의 Webhook URL을 기입
    channel: # 기본 설정한 채널이 아닌 다른 채널로 보내고 싶을 때 기입한다
    pretext: # attachments의 상단에 나오게 되는 일반 텍스트 문자
    color: # attachment에 왼쪽 사이드 컬러. default=red
    author-name: # attachment의 상단에 나오는 이름
    author-icon: # author-icon 왼쪽에 나올 아이콘의 url링크
    footer: # attachment에 하단에 나올 부분. default=현재 시간

##시큐리티 인증 uri관리
authorization-uri:
  denyUrls:
    - /github/users
    - /github/my-rank
    - /github/open
    - /boj/my-rank
    - /boj/users
    - /job/**
  ##actuator 설정
management:
  endpoints:
    web:
      exposure:
        include: prometheus,refresh,health,metrics

```

