포팅메뉴얼

1. 사용한 JVM, 웹서버, WAS, 제품 등 종류와 설정 값, 버전
   - Build Tool : Gradle
   - Back-End 언어 : Java 11
   - Crawling 언어 : Python3.x
   - FrameWork : SpringBoot 2.7.17
   - Font-End : React 18.2.0
   - DB : Redis 7.2.1, MongoDB Cloud, MySQL 8.0, ElasticSearcj 7.4.2 (ELK)
   - IFA : Ubuntu 20.04, Docker, Jenkins

2. 로그인 정보
   - Jenkins

```
id : admin
pw : ssafy@b205@

http://k9b205.p.ssafy.io:9090/login?from=%2F
```

   - MySQL

```
url: jdbc:mysql://k9b205.p.ssafy.io
username: b205
password: 9gi_ssafy_final
```

   - MongoDB

```
mongodb+srv://S09P31B205:z5HxUpl4gB@ssafy.ngivl.mongodb.net/S09B31B205?authSource=admin
```

▼ spring boot application.yml

```
server:  port: [port번호]
    tomcat:
        accept-count: 600
        connection-timeout: 5000
    threads:
        max: 500
        min-spare: 200

    url:
        host: [url 주소]
    spring:
        jackson:
            time-zone: Asia/Seoul
            data:
```

```
    mongodb:
      uri: [mongodb 주소]
  datasource:
    driver-class-name: [sql db 주소]
    username: [계정 이름]
    password: [비밀번호]
  jpa:
    open-in-view: false
    properties:
      hibernate:
        show_sql: true
        format_sql: true
        use_sql_comments: true
        ddl-auto: none
  logging:
    level:
      org.hibernate.SQL: debug
      org.hibernate.type: trace
  mail:
    host: smtp.gmail.com
    port: 587
    username: [메일 주소]
    password: [메일 비밀번호]
    properties:
      mail:
        smtp:
          auth: true
          starttls:
            enable: true
    redis:
      host: [레디스 host 주소]
      port: [레디스 포트번호]
      password: [레디스 비밀번호]
  jwt:
    secretKey: [jwt 시크릿 키 번호]
  host:
    url: http://localhost:8080
```

▼ docker-compose.yml

```
version: '3'

services:
```

```
    jenkins:
        restart: always # 컨테이너 다운 시 재시작하라는 명령어
        image: jenkins/jenkins:lts
        container_name: jenkins
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
            - /jenkins:/var/jenkins_home
        ports:
            - "9090:8080"
        user: root
        environment:
            - TZ=Asia/Seoul
        #   - JENKINS_OPTS="--prefix=/jenkins"
    mysql:
        restart: always # 컨테이너 다운 시 재시작하라는 명령어
        image: mysql:8.0 # 컨테이너에서 사용하는 base image 지정
        container_name: mysql
        ports: # -p 옵션과 동일
            - 4000:3306
        environment: # 컨테이너 안의 환경변수 설정
            MYSQL_ROOT_PASSWORD: 9gi_ssafy_final
        command:
            - --character-set-server=utf8mb4
            - --collation-server=utf8mb4_unicode_ci
        volumes:
            - ./mysqldata:/var/lib/mysql
    redis:
        restart: always # 컨테이너 다운 시 재시작하라는 명령어
        image: redis
        container_name: redis
        ports:
            - 6379:6379
        command: redis-server --requirepass 9gi_ssafy_final
networks:
    front:
    back:
```

▼ nginx.conf

/etc/nginx/nginx.conf

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {
        upstream front {
                server localhost:3000;
        }

        upstream back {
                server localhost:8080;
        }

        upstream elastic {
                server localhost:8081;
        }

        ##
        # Basic Settings
        ##

        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;
        # server_tokens off;
```

```
        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

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

        ##
        # SSL Settings
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
        ssl_prefer_server_ciphers on;

        ##
        # Logging Settings
        ##

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        ##
        # Gzip Settings
        ##

        gzip on;

        # gzip_vary on;
        # gzip_proxied any;
        # gzip_comp_level 6;
        # gzip_buffers 16 8k;
        # gzip_http_version 1.1;
        # gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text

        ##
        # Virtual Host Configs
        ##

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;

        server {
                listen 443 ssl;
                ssl_certificate /etc/letsencrypt/live/k9b205.p.ssafy.io/fullchain.pem;
                ssl_certificate_key /etc/letsencrypt/live/k9b205.p.ssafy.io/privkey.pem;
                ssl_protocols TLSv1.2 TLSv1.3;
                ssl_ciphers 'TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:ECDHE-RSA-AES128-GCM-SHA256';

                # 이 웹서버에 어떤 방식으로 들어왔는지 확인
                server_name k9b205.p.ssafy.io;

                location ~ ^/(api|docs|actuator) {
                        proxy_pass      http://back;
#                       add_header 'Access-Control-Allow-Origin' '*' always;
                        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always;
                        add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control
                        add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
                }

                location /elastic {
                        proxy_pass      http://elastic;
                        add_header 'Access-Control-Allow-Origin' '*' always;
                        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always;
                        add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control
                        add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
                }

                # / 경로로 오는 요청을 프론트엔드 upstream의 /경로로 포워딩
                # 실제 사용자는 front로 접근하지 않았지만 그와 동일한 효과가 발생한다
                location / {
                        proxy_pass      http://front;
#                       add_header 'Access-Control-Allow-Origin' '*' always;
                        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always;
                        add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control
                        add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
                }

                # 다른 경로로 접근해서 404 발생 시 리다이렉팅
                error_page 404 = @notfound;
                location @notfound {
                        return 302 /;
                }
        }

        # redirect to port 443 when signal comes to port 80
```

```
        server {
                listen 80;
                return 301 https://$host$request_uri;

                server_name k9b205.p.ssafy.io;
                return 404;
        }

        # redirect to domain when signal comes to ip address
        server {
                listen 80;
                listen 443 ssl;

                ssl_certificate /etc/letsencrypt/live/k9b205.p.ssafy.io/fullchain.pem;
                ssl_certificate_key /etc/letsencrypt/live/k9b205.p.ssafy.io/privkey.pem;

                server_name 15.165.17.14;
                return 301 https://k9b205.p.ssafy.io;
        }
}


#mail {
#       # See sample authentication script at:
#       # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript
#
#       # auth_http localhost/auth.php;
#       # pop3_capabilities "TOP" "USER";
#       # imap_capabilities "IMAP4rev1" "UIDPLUS";
#
#       server {
#               listen     localhost:110;
#               protocol   pop3;
#               proxy      on;
#       }
#
#       server {
#               listen     localhost:143;
#               protocol   imap;
#               proxy      on;
#       }
#}
```

▼ api.conf

/etc/nginx/sites-available/api.conf

```
server {
        listen 443 ssl;
        ssl_certificate /etc/letsencrypt/live/api4u.site/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/api4u.site/privkey.pem;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers 'TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:ECDHE-RSA-AES128-GCM-SHA256';

        # 이 웹서버에 어떤 방식으로 들어왔는지 확인
        server_name api4u.site www.api4u.site;

        location ~ ^/(api|docs|actuator) {
#               limit_req zone=mylimit burst=20;
#               limit_req_status 429;
                proxy_pass      http://back;
#               add_header 'Access-Control-Allow-Origin' '*' always;
                add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always;
                add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content
                add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
        }

        location /elastic {
                proxy_pass      http://elastic;
                add_header 'Access-Control-Allow-Origin' '*' always;
                add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always;
                add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content
                add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
        }
```

```
        # / 경로로 오는 요청을 프론트엔드 upstream의 /경로로 포워딩
        # 실제 사용자는 front로 접근하지 않았지만 그와 동일한 효과가 발생한다
        location / {
                proxy_pass      http://front;
                add_header 'Access-Control-Allow-Origin' '*' always;
                add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always;
                add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content
                add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
        }

        # 다른 경로로 접근해서 404 발생 시 리다이렉팅
        error_page 404 = @notfound;
        location @notfound {
                return 302 /;
        }
}

# redirect to port 443 when signal comes to port 80
server {
        listen 80;
        return 301 https://$host$request_uri;

        server_name api4u.site www.api4u.site;
        return 404;
}

# redirect to domain when signal comes to ip address
server {
        listen 80;
        listen 443 ssl;

        ssl_certificate /etc/letsencrypt/live/api4u.site/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/api4u.site/privkey.pem;

        server_name 15.165.17.14;
        return 301 https://api4u.site;
}
```

▼ 백엔드 파이프라인

```
pipeline {
    agent any     // 사용 가능한 에이전트에서 이 파이프라인 또는 해당 단계를 실행

    environment {
        GIT_URL = "https://lab.ssafy.com/s09-final/S09P31B205.git"
        BLUE_CONTAINER = "back-blue"
        GREEN_CONTAINER = "back-green"
        CONTAINER_NAME = "back-container"
        IMAGE_NAME = "back-image"
    }

    stages {
        stage('Git clone') {
            steps {
                sh 'echo "Cloning repository."'
                git branch: 'develop-back',
                url: "${GIT_URL}",
                credentialsId: "3c4da37d-5a4b-4276-a5af-7164a547e160"
            }

            post {
                success {
                    sh 'echo "Successfully Cloned Repository"'
                }
                failure {
                    sh 'echo "Fail Cloned Repository"'
                }
            }
        }

        stage('Build') {
            steps {
                // gralew이 있어야됨. git clone해서 project를 가져옴.
                sh '''
                    cd backend
```

```
                chmod +x ./gradlew
                ./gradlew clean build
            '''
        }
        post {
            success {
                echo 'gradle build success'
            }

            failure {
                echo 'gradle build failed'
            }
        }
    }

    stage('Test') {
        steps {
            echo '백엔드 프로젝트의 테스트 코드를 실행합니다.'
            sh '''
                cd backend
                ./gradlew test
            '''
        }

        post {
            success {
                sh 'echo "test Success"'
            }
            failure {
                sh 'echo "test Fail"'
            }
        }
    }

    stage('Docker delete') {
        steps {
            script {
                try {
                    // 컨테이너가 존재하면 삭제합니다.
                    sh "docker stop ${CONTAINER_NAME}"
                    sh "docker rm -f ${CONTAINER_NAME}"
                } catch (Exception e) {
                    // 컨테이너가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.
                    echo "Docker container ${CONTAINER_NAME} does not exist. Skipping deletion."
                }

                try {
                    // 이미지가 존재하면 삭제합니다.
                    sh "docker image rm ${IMAGE_NAME}"
                } catch (Exception e) {
                    // 이미지가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.
                    echo "Docker image ${IMAGE_NAME} does not exist. Skipping deletion."
                }
            }
        }


        post {
            success {
                sh 'echo "docker delete Success"'
            }
            failure {
                sh 'echo "docker delete Fail"'
            }
        }
    }

    stage('Dockerizing'){
        steps{
            sh 'echo " Image Bulid Start"'
            sh """
                cd backend
                docker build -t ${IMAGE_NAME} .
            """
        }
        post {
            success {
                sh 'echo "Bulid Docker Image Success"'
            }

            failure {
                sh 'echo "Bulid Docker Image Fail"'
            }
        }
```

```
        }

        stage('Deploy') {
            steps {
                // script {
                //     def blueRunning = sh(script: "docker ps --format '{{.Names}}' | grep -w ${BLUE_CONTAINER}", returnStatus: t

                //     if (blueRunning) {
                //         TARGET_CONTAINER = GREEN_CONTAINER // 블루가 실행중일때
                //     } else {
                //         TARGET_CONTAINER = BLUE_CONTAINER // 블루가 실행중이 아닐때
                //     }

                //     if (TARGET_CONTAINER == GREEN_CONTAINER) {
                //         sh """
                //             docker run --name ${GREEN_CONTAINER} -d -p 8080:8080 ${IMAGE_NAME}
                //         """
                //     } else {
                //         sh """
                //             docker run --name ${BLUE_CONTAINER} -d  -p 7080:8080 ${IMAGE_NAME}
                //         """
                //     }

                //     def myNumbers = [1, 2, 3, 4, 5]
                //     for (retry_count in myNumbers) {
                //         if (sh(script: "docker ps -q -f name=${TARGET_CONTAINER}", returnStatus: true) == 0) {
                //             echo "✅ Health Checking 에 성공했습니다!"
                //             break
                //         }

                //         if (retry_count == 5) {
                //             echo "❌ Health checking 에 실패했습니다."
                //             error("Health checking 실패")
                //         }

                //         echo "🏥 10초후에 다시 Health Checking 이 시도될 예정입니다."
                //         sleep 10
                //     }

                //     if (TARGET_CONTAINER == GREEN_CONTAINER) {
                //         sh "echo 'sudo set \$develop-back https://k9b205.p.ssafy.io:8080;' > /etc/nginx/back-url.inc"
                //         echo "Switch the reverse proxy direction of nginx to ${TARGET_CONTAINER} : 8080 🔄"
                //         sh "docker exec nginx nginx -s reload"
                //         try {
                //             // 컨테이너가 존재하면 삭제합니다.
                //             sh "docker stop ${BLUE_CONTAINER}"
                //             sh "docker rm -f ${BLUE_CONTAINER}"
                //         } catch (Exception e) {
                //             // 컨테이너가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.
                //             echo "Docker container ${BLUE_CONTAINER} does not exist. Skipping deletion."
                //         }
                //     } else {
                //         sh "echo 'sudo set \$develop-back https://k9b205.p.ssafy.io:8080;' > /etc/nginx/back-url.inc"
                //         echo "Switch the reverse proxy direction of nginx to ${TARGET_CONTAINER} : 7080 🔄"
                //         sh "docker exec nginx nginx -s reload"
                //         try {
                //             // 컨테이너가 존재하면 삭제합니다.
                //             sh "docker stop ${GREEN_CONTAINER}"
                //             sh "docker rm -f ${GREEN_CONTAINER}"
                //         } catch (Exception e) {
                //             // 컨테이너가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.
                //             echo "Docker container ${GREEN_CONTAINER} does not exist. Skipping deletion."
                //         }
                //     }
                // }

                sh "docker run --name ${CONTAINER_NAME} -d -p 127.0.0.1:8080:8080 ${IMAGE_NAME}"
            }

            post {
                success {
                    echo 'deploy success'
                }

                failure {
                    echo 'deploy failed'
                }
            }
        }
    }
}
```

▼ 프론트엔드 파이프라인

```
pipeline {
    agent any     // 사용 가능한 에이전트에서 이 파이프라인 또는 해당 단계를 실행

    environment {
        GIT_URL = "https://lab.ssafy.com/s09-final/S09P31B205.git"
        CONTAINER_NAME = "front-container"
        IMAGE_NAME = "front-image"
    }

    stages {
        stage('Git clone') {
            steps {
                git branch: 'develop-Front', //BE 브랜치 가져오기
                    url: "${GIT_URL}",
                    credentialsId: "3c4da37d-5a4b-4276-a5af-7164a547e160"
            }

            //클론 성공실패유무에 따라 echo 실행
            post {
                success {
                    sh 'echo "Successfully Cloned Repository"'
                }
                failure {
                    sh 'echo "Fail Cloned Repository"'
                }
                     }
            }   //end of stage

        stage('Docker delete') {
            steps {
                script {
                    try {
                        // 컨테이너가 존재하면 삭제합니다.
                        sh "docker stop ${CONTAINER_NAME}"
                        sh "docker rm -f ${CONTAINER_NAME}"
                    } catch (Exception e) {
                        // 컨테이너가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.
                        echo "Docker container ${CONTAINER_NAME} does not exist. Skipping deletion."
                    }

                    try {
                        // 이미지가 존재하면 삭제합니다.
                        sh "docker image rm ${IMAGE_NAME}"
                    } catch (Exception e) {
                        // 이미지가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.
                        echo "Docker image ${IMAGE_NAME} does not exist. Skipping deletion."
                    }
                }
            }


            post {
                success {
                    sh 'echo "docker delete Success"'
                }
                failure {
                    sh 'echo "docker delete Fail"'
                }
            }
        }

        stage('Dockerizing'){
            steps{
                sh 'echo " Image Bulid Start"'
                sh """
                    cd front-end
                    docker build -t ${IMAGE_NAME} .
                """
            }
            post {
                success {
                    sh 'echo "Bulid Docker Image Success"'
                }

                failure {
                    sh 'echo "Bulid Docker Image Fail"'
                }
            }
```

```
        }

        stage('Deploy') {
            steps {
                //내부 localhost 통신만을 위해 외부 포트를 제한하였다.
                sh "docker run --name ${CONTAINER_NAME} -d -p 127.0.0.1:3000:3000 ${IMAGE_NAME}"
            }

            post {
                success {
                    echo 'deploy success'
                }

                failure {
                    echo 'deploy failed'
                }
            }
        }

    }
}
```

▼ 백엔드 Dockerfile

```
# 기본 이미지로 Java 11을 사용합니다..
FROM openjdk:11-jre-slim

# 작업 디렉토리를 설정합니다.
WORKDIR /app

# 호스트 머신에서 JAR 파일을 복사합니다.
COPY build/libs/*.jar app.jar

# JAR 파일 실행
CMD ["java", "-jar", "app.jar"]
```

▼ 프론트엔드 Dockerfile

```
# 개발 서버 실행 환경
# FROM: 이미지 지정
FROM node:18

# WORKDIR: RUN, CMD, ENTRYPOINT, ADD, COPY 에 정의된 명령어를 실행하는 작업 디렉터리 지정
WORKDIR /app

# COPY: 이미지에 파일이나 폴더를 추가
COPY package.json .

# RUN: 이미지를 빌드하며 실행할 명령어 지정
RUN npm install

# COPY: 이미지에 파일이나 폴더를 추가
COPY . .

# EXPOSE: 이미지가 통신에 사용할 포트를 명시적으로 지정
EXPOSE 3000

# CMD: 컨테이너를 실행할 때 실행할 명령어 지정
CMD ["npm", "start"]
```