포팅 매뉴얼

목차

해당 글은 (https://zesty-pheasant-3d2.notion.site/2b13c50dd6964281be806f32d1bbe33d?pvs=4) 에서 확인할 수 있습니다.

1. 개요 및 목적

혼자서 운동을 하기에는 **과도한 기대와 급한 목표 설정, 습관의 부재, 부족한 지원 시스템** 등의 이유로 운동을 포기하게 됩니다.

다른 사람들과의 소통을 통해 운동을 할 때 동기부여를 받고 동시에 도움도 받을 수 있는 SNS를 기획하였습니다.

2. 개발 환경

🖥️ Frontend

- Visual Studio Code 1.85.1
- Node.js 20.10.0
- ReactNative
- Expo

📋 Backend

- IntelliJ 2023.3.2 (Ultimate Edition)
- Java 17.0.9
- Spring Boot 3.1.7

📚 DB

- MySQL 8.3.0
- Redis

🌏 Deploy

- AWS EC2 Ubuntu 20.04.6 LTS
- Jenkins 2.426.2
- Docker 25.0.0
- Nginx 1.25.3

📣 Communication

- 형상 관리 - `Gitlab`
- 이슈 및 스크럼 관리 - `Jira`
- 의사소통, 협업 - `Notion`, `Mattermost`
- 디자인 - `Figma`

3. EC2 서버 설정

1. Jenkins

1. docker가 설치된 jenkins 이미지 만들기

`docker-install.sh` 파일을 먼저 생성 후 아래의 코드 입력합니다.

```
#!/bin/sh
apt-get update && \
apt-get -y install apt-transport-https \
     ca-certificates \
     curl \
     gnupg2 \
     zip \
     unzip \
     software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-rele
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/$(. /e
   $(lsb_release -cs) \
   stable" && \
apt-get update && \
apt-get -y install docker-ce
```

`dockerfile` 생성해줍니다

```
FROM jenkins/jenkins:lts

#root 계정으로 변경(for docker install)
USER root

COPY docker-install.sh /docker-install.sh
RUN chmod +x /docker-install.sh
RUN /docker-install.sh
```

```
RUN usermod -aG docker jenkins
USER jenkins
```

위와 같이 `docker-install.sh`와 `dockerfile`을 만든후 아래의 명령어로 image 생성합니다.

```
docker build -t jenkins-docker .
```

2. Front Pipeline Script

본 프로젝트는 `front`, `back` 2개의 브랜치로 나누어 관리하기 때문에 2개의 파이프라인에 각각의 script를 작성해야 합니다.

우선 Front부분 파이프라인 script 입니다.

`Front pipeline Script`

```
pipeline {
    agent any

    tools {
        nodejs "20.10.0"
    }

    environment {
        repository = ${docker repository} // Docker 이미지의 저
        dockerImage = ${docker image} // Docker 이미지 변수 초기

        registryCredential = ${docker hub credential} // Dock

        releaseServerAccount = ${server account} // ssh Accou
        releaseServerUri = ${server uri} // ssh URI
    }

    stages {
        stage('Git Clone') {
            steps {
```

```
                git branch: ${front branch name}, // clone 받을
                credentialsId: ${gitlab credential}, // GitLa
                url: ${gitlab url} // GitLab Clone 경로
            }
        }
        stage('Image Build & DockerHub Push') {
            steps {
                dir('frontend') {
                    script {
                        try{
                            docker.withRegistry('', registryC
                                sh "docker buildx create --us
                                sh "docker buildx build --pla
                                sh "docker buildx build --pla
                            }
                            sendMattermostNotification("☑️ FR
                        } catch (Exception e){
                            sendMattermostNotification("⛔ FR
                            throw e
                        }
                    }
                }

            }
        }
        stage('Before Service Stop') {
            steps {
                script{
                    try{
                        sshagent(credentials: ['ubuntu_jenkin
                            /* 현재 돌아가고 있는 API 서버가 있다면
                            sh '''
                                if
                                ssh -o StrictHostKeyChecking=
                                then
                                    ssh -o StrictHostKeyCheck
                                    ssh -o StrictHostKeyCheck
                                    docker rmi $repository:la
```

```
                            fi
                        '''
                    }
                    sendMattermostNotification("♻️ FRONTE
                } catch (Exception e) {
                    sendMattermostNotification("⛔ FRONTE
                    throw e
                }
            }
        }
    }
    stage('DockerHub Pull') {
        steps {
                sh "docker pull $repository:latest" // 최신
        }
    }
    stage('Service Start') {
        steps {
            sshagent(credentials: ['ubuntu_jenkins']) {
                /* 최신버전 실행 */
                sh '''
                    ssh -o StrictHostKeyChecking=no $rele
                '''
            }
        }
    }
    stage('Service Check') {
        steps {
            sshagent(credentials: ['ubuntu_jenkins']) {
                /* Health Check : 20번 동안 연결하여 상태코드를
                script {
                    for (int retry_count = 1; retry_count
                        try {
                            // 원격 서버에서 curl 명령을 실행하
                            def sshOutput = sh(script: "s
                            if (sshOutput.contains('200')
                                sendMattermostNotificatio
                                break
```

```
                                }

                                if (retry_count == 20) {
                                    sendMattermostNotificatio
                                    throw new Exception("Serv
                                }

                                echo "The server is not alive
                                sleep(5)
                            } catch (Exception e) {
                                sendMattermostNotification("
                                throw e
                            }
                        }
                    }
                }
            }
        }
    }
}


/* MatterMost 알리미*/
def sendMattermostNotification(String title, String message,
    def fullMessage = "## $title \nBuildNumber : ${currentBui

    /* Error Message 가 존재한다면 추가 */
    if (errorMessage) {
        fullMessage += "\nError Message : $errorMessage"
    }

    mattermostSend(
        message: fullMessage,
        color: color,
        channel: "A605_"
    )
}
```

`Dockerfile` 작성

```
FROM node:20.10.0-slim

WORKDIR /usr/src/app

COPY ./ ./

RUN npm install

CMD npx expo start
```

3. Back Pipeline Script

다음은 Back 부분 파이프라인 script 입니다.

`Back pipeline script`

```
pipeline {
    agent any

    environment {
        repository = ${docker repository} // Docker 이미지의 저
        dockerImage = ${docker image} // Docker 이미지 변수 초기

        registryCredential = ${docker hub credential} // Dock

        releaseServerAccount = ${server account} // ssh Accou
        releaseServerUri = ${server uri} // ssh URI
    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: ${back branch name}, // clone 받을
                credentialsId: ${gitlab credential}, // GitLa
                url: ${gitlab url} // GitLab Clone 경로
```

```
            }
        }
        stage('Jar Build') {
            steps {
                script{
                    try{
                        dir('backend/Igemoji'){
                            sh 'chmod +x ./gradlew' // gradle
                            sh './gradlew clean bootJar' // G
                        }
                        sendMattermostNotification("☑️ BACKEN
                    } catch (Exception e){
                        sendMattermostNotification("⛔ BACKEN
                        throw e
                    }
                }
            }
        }
        stage('Image Build & DockerHub Push') {
            steps {
                dir('backend/Igemoji') {
                    script {
                        docker.withRegistry('', registryCrede
                            sh "docker buildx create --use --
                            sh "docker buildx build --platfor
                            sh "docker buildx build --platfor
                        }
                    }
                }

            }
        }
        stage('Before Service Stop') {
            steps {
                script{
                    try{
                        sshagent(credentials: ['ubuntu_jenkin
                            /* 현재 돌아가고 있는 API 서버가 있다면
```

```
                        sh '''
                            if
                            ssh -o StrictHostKeyChecking=
                            then
                                ssh -o StrictHostKeyCheck
                                ssh -o StrictHostKeyCheck
                                docker rmi $repository:la
                            fi
                        '''
                    }
                    sendMattermostNotification("♻️ BACKEN
                } catch (Exception e) {
                    sendMattermostNotification("⛔ BACKEN
                    throw e
                }
            }
        }
    }
    stage('DockerHub Pull') {
        steps {
                sh "docker pull $repository:latest" // 최신
        }
    }
    stage('Service Start') {
        steps {
            sshagent(credentials: ['ubuntu_jenkins']) {
                /* 최신버전 실행 */
                sh '''
                    ssh -o StrictHostKeyChecking=no $rele
                '''
            }
        }
    }
    stage('Service Check') {
        steps {
            sshagent(credentials: ['ubuntu_jenkins']) {
                /* Health Check : 20번 동안 연결하여 상태코드를
                script {
```

```
                        for (int retry_count = 1; retry_count
                            try {
                                // 원격 서버에서 curl 명령을 실행ㅎ
                                def sshOutput = sh(script: "s
                                if (sshOutput.contains('200')
                                    sendMattermostNotificatio
                                    break
                                }

                                if (retry_count == 20) {
                                    sendMattermostNotificatio
                                    throw new Exception("Serv
                                }

                                echo "The server is not alive
                                sleep(5)
                            } catch (Exception e) {
                                sendMattermostNotification("
                                throw e
                            }
                        }
                    }
                }
            }
        }
    }
}


/* MatterMost 알리미*/
def sendMattermostNotification(String title, String message,
    def fullMessage = "## $title \nBuildNumber : ${currentBui

    /* Error Message 가 존재한다면 추가 */
    if (errorMessage) {
        fullMessage += "\nError Message : $errorMessage"
    }
```

```
    mattermostSend(
        message: fullMessage,
        color: color,
        channel: "A605_"
    )
}
```

`dockerfile` 작성

```
FROM openjdk:17
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

4. Back 환경변수 설정

추가로 spring의 application.properties를 설정해줍니다.

```
spring:
  # MYSQL
  datasource:
    url: jdbc:mysql://${MYSQL_HOST}:${MYSQL_PORT}/${MYSQL_DAT
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: ${MYSQL_USER}
    password: ${MYSQL_PASSWORD}

  # Redis
  data:
    redis:
      host: ${REDIS_HOST}
      port: ${REDIS_PORT}


  # JPA
  jpa:
    hibernate:
      ddl-auto: update
```

```
    properties:
      hibernate:
        format_sql: true
    show-sql: true

# LOG
logging:
  level:
    org:
      springframework:
        web:
          socket: INFO

# OAUTH2
oauth2:
  client:
    registration:
      kakao:
        client-id: ${KAKAO_CLIENT_ID}
        client-secret: ${KAKAO_CLIENT_SECRET}
        authorization-grant-type: authorization_code
        redirect-uri: ${KAKAO_REDIRECT_URI}
        client-authentication-method: POST
    provider:
      kakao:
        authorization-uri: https://kauth.kakao.com/oauth/auth
        token-uri: https://kauth.kakao.com/oauth/token
        user-info-uri: https://kapi.kakao.com/v2/user/me
        user-name-attribute: id

# JWT Configuration
jwt:
  secret: ${JWT_SECRET}

# server port
server:
  port: ${SERVER_PORT}
```

```
# AWS S3
cloud:
  aws:
    credentials:
      access-key: ${S3_ACCESS_KEY}
      secret-key: ${S3_SECRET_KEY}
    s3:
      bucket: igemoji
    region:
      static: ap-northeast-2
```

2. Nginx

1. SSL 인증서 발급

Certbot 설치

```
sudo apt-get install certbot
```

SSL 인증서 발급

```
sudo certbot certonly --manual --preferred-challenges dns -d
```

명령어를 실행하면 나오는 _acme-challenge을 가비아 DNS 설정에서 다음과 같이 입력해주면 됩니다.

DNS 설정 레코드 수정

타입	호스트	값/위치	TTL
A	@	[redacted]	3600
A	back	[redacted]	3600
A	jenkins	[redacted]	3600
TXT	_acme-challenge	"[redacted]"	600
TXT	_acme-challenge	"[redacted]"	600
A	www	[redacted]	3600

2. Nginx 설정

nginx 설정 파일 작성

- `/etc/nginx/conf.d` 경로에 `defaut.conf` 설정 파일을 만들어 아래의 코드를 입력합니다

```
server {
    listen 80;
    server_name igemoji.store;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name jenkins.igemoji.store;

    ssl_certificate /etc/letsencrypt/live/igemoji.store/fullc
    ssl_certificate_key /etc/letsencrypt/live/igemoji.store/p

    location / {
        proxy_pass http://jenkins:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_http_version 1.1;
```

```
        proxy_request_buffering off;
        proxy_buffering off;
        add_header 'X-SSH-Endpoint' 'jenkins.igemoji.store' a
    }
}

server {
    listen 443 ssl;
    server_name www.igemoji.store;

    ssl_certificate /etc/letsencrypt/live/igemoji.store/fullc
    ssl_certificate_key /etc/letsencrypt/live/igemoji.store/p

    location / {
        proxy_pass http://igemoji-front:8081;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_buffering off;
    }
}

server {
    listen 443 ssl;
    server_name back.igemoji.store;

    ssl_certificate /etc/letsencrypt/live/igemoji.store/fullc
    ssl_certificate_key /etc/letsencrypt/live/igemoji.store/p

    location / {
        proxy_pass http://igemoji-back:8881;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_http_version 1.1;
```

```
        proxy_request_buffering off;
        proxy_buffering off;

        limit_except GET POST PUT DELETE {
            allow all;
        }
    }

    location /similar {
        proxy_pass http://igemoji-similar:8000;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_buffering off;
    }

    location /ws {
        proxy_pass http://igemoji-back:8881;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_http_version     1.1;
        proxy_set_header      Upgrade $http_upgrade;
        proxy_set_header      Connection "upgrade";
        proxy_read_timeout    86400;
    }
}
```

3. Docker-compose

마지막으로 `Docker-compose.yml` 파일을 생성 후 아래의 코드를 입력합니다.

```
version: '3'
services:
  jenkins:
    image: jenkins-docker
```

```
    volumes:
      - /home/ubuntu/jenkins:/var/jenkins_home # jenkins가 돌아
      - /home/ubuntu/.ssh:/var/jenkins_home/.ssh # 호스트 ssh 키
      - /var/run/docker.sock:/var/run/docker.sock # host의 doc
    networks:
      - nat

  nginx:
    image: nginx

    ports:
      - 80:80
      - 443:443
    volumes:
      - /home/ubuntu/nginx/conf.d:/etc/nginx/conf.d #
      - /home/ubuntu/nginx/cert:/etc/cert # 인증서 파일을 공유하기
      - /etc/letsencrypt:/etc/letsencrypt
    restart: always # 꺼져도 다시 실행
    depends_on: # jenkins가 실행된 후에 nginx 실행
      - jenkins
    networks:
      - nat

  igemoji-back:
    image: blank98/igemoji-back:latest

    env_file:
      - ./back.env
    depends_on:
      - db
    networks:
      - nat

  igemoji-similar:
    image: blank98/igemoji-similar:latest

    networks:
```

```
      - nat

  igemoji-front:
    image: blank98/igemoji-front:latest

    networks:
      - nat

  db:
    image: mysql:latest

    ports:
      - 3300:3306

    volumes:
      - mysql:/home/ubuntu/mysql
    environment:
      MYSQL_DATABASE: igemoji
      MYSQL_USER: a605
      MYSQL_PASSWORD: ssafyA605
      MYSQL_ROOT_PASSWORD: ssafyA605
    networks:
      - nat

  redis:
    image: redis:latest
    ports:
      - 6379:6379
    volumes:
      - redis-data:/data

networks:
  nat:
    external: true

volumes:
  mysql:
  redis-data:
```

4. 빌드하기

위의 모든 과정을 완료하였다면 서비스를 실행할 수 있습니다.

```
docker-compose up
```