포팅메뉴얼

1. 개요

요가요 프로젝트는 **Spring Boot, MySQL, Android Studio**를 기반으로 개발되었습니다. **Docker**와 **Jenkins**를 사용하였으며 **스프링 서버, MySQL, 젠킨스**를 각각 컨테이너에 담아 배포하였습니다. 본 문서는 프로젝트를 배포하고 운영하는 방법을 설명합니다.

1.1. 프로젝트 사용 도구

이슈 관리 : JIRA
형상 관리 : Gitlab
커뮤니케이션 : Notion, Mattermost
디자인 : Figma
UCC : 모바비
CI/CD : Jenkins

1.2. 개발 환경

#Frontend
Android Studio : Android Studio Ladybug
Android SDK : 34
IntelliJ : IntelliJ IDEA 2024.3.1.1

#Backend
Java 17
SpringBoot 3.4.3
JPA
MySQL

#Infra
Docker
Docker Compose
Jenkins

```
Nginx

#AI
Python
Tensorflow
OpenCV
MediaPipe
```

1.3. 외부 서비스

```
AWS S3 Stroage (과금이 발생할 수 있습니다 취급 주의)
Coturn
```

2. 환경 설정

2.1. EC2에 필요한 프로그램 설치

EC2 서버에서 배포에 필요한 프로그램들을 설치합니다.

```
#java 설치
sudo apt update
sudo apt install openjdk-21-jdk -y // 자바 설치

#Docker 설치
sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker // 도커 설치

#Docker-compose 설치
sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

#git 설치
sudo yum install git -y
```

```
#Nginx 설치
sudo apt install nginx -y
```

2.2. EC2 방화벽 설정

```
#포트 허용
sudo ufw allow 80 // HTTP
sudo ufw allow 443 // HTTPS
sudo ufw allow 8080 // SpringBoot
sudo ufw allow 18080 // Jenkins
```

3. 배포 파일

3.1. DockerFile

```
FROM openjdk:17-jdk-slim

COPY YogaYoBack-0.0.1-SNAPSHOT.jar yogayo.jar

EXPOSE 8080

ENV SPRING_PROFILES_ACTIVE=prod

CMD ["java", "-jar", "/yogayo.jar"]
```

3.2. Docker Compose 파일

```
services:
  spring-app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: yogayo_container // 스프링 컨테이너
    image: yogayo_image
    ports:
      - "8080:8080"
    environment:
```

```
    - SPRING_DATASOURCE_URL=${DB_PROD_URL}
    - SPRING_DATASOURCE_USERNAME=${DB_PROD_USERNAME}
    - SPRING_DATASOURCE_PASSWORD=${DB_PROD_PASSWORD}
    - your-bucket-name=${your-bucket-name}
    - yourAccessKey=${yourAccessKey}
    - yourSecretKey=${yourSecretKey}
  depends_on:
    - mysql
  command: ["java", "-jar", "/yogayo.jar"]
  networks:
    - app-network

mysql:
  image: mysql:8
  container_name: mysql // mysql 컨테이너
  environment:
    MYSQL_ROOT_PASSWORD: ${DB_PROD_PASSWORD}
    MYSQL_DATABASE: yogayo
  ports:
    - "3306:3306"
  networks:
    - app-network

jenkins:
  image: jenkins/jenkins:lts  // 젠킨스 컨테이너
  container_name: jenkins
  privileged: true
  user: root
  environment:
    - DOCKER_HOST=unix:///var/run/docker.sock
  ports:
    - "18080:8080"
    - "50000:50000"
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - ./jenkins_home:/var/jenkins_home
    - /usr/bin/docker:/usr/bin/docker
  networks:
```

```
    - app-network

networks:  // 네트워크 설정
  app-network:
    driver: bridge
```

3.3. Jenkins 파일

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE = credentials('DOCKER_IMAGE') // 빌드해서 만든 이미지명
        DOCKER_CONTAINER = credentials('DOCKER_CONTAINER') // 컨테이너명
        DOCKER_PORT = credentials('DOCKER_PORT')
        DOCKER_PATH = '/home/ubuntu/yoga-docker' // EC2 서버의 Docker 저장

        EC2_USER = credentials('EC2_USER')
        EC2_IP = credentials('EC2_IP')
        SSH_KEY = credentials('ssafyd104')
        SPRING_PROFILES_ACTIVE = 'prod'

        DB_PROD_URL = credentials('DB_PROD_URL')
        DB_PROD_USERNAME = credentials('DB_PROD_USERNAME')
        DB_PROD_PASSWORD = credentials('DB_PROD_PASSWORD')

        S3_BUCKET_NAME = credentials('your-bucket-name')
        S3_ACCESS_KEY = credentials('yourAccessKey')
        S3_SECRET_KEY = credentials('yourSecretKey')

        JWT_SECRET_KEY = credentials('JWT_SECRET_KEY')

        JAVA_HOME = '/opt/java/openjdk'
        GRADLE_HOME = '/var/jenkins_home/gradle-8.12.1'
        PATH = "${JAVA_HOME}/bin:${GRADLE_HOME}/bin:${env.PATH}"
    }

    tools {// 젠킨스에 등록해둔 이름으로...
```

```
        jdk 'jdk17'
        gradle 'Gradle-8.12.1'
    }

    stages {
        stage('Clone Repository') {
            steps {
                echo 'Cloning the repository...'
                git branch: 'develop-back',
                    url: 'https://lab.ssafy.com/s12-ai-image-sub1/S12P21D104.git',
                    credentialsId: 'GITLAB_PAT'
            }
        }

        stage('Build Application') {
            steps {
                echo 'Building the application with Gradle Wrapper...'
                dir('yogaback') {
                    sh 'gradle clean build -x test'
                    sh 'ls -al $(pwd)/build/libs'
                }
            }
        }
        stage('Build Docker Image') {
            steps {
                echo 'Building the Docker image...'
                dir('yogaback') {
                    sh 'cp build/libs/YogaYoBack-0.0.1-SNAPSHOT.jar .'
                    sh 'docker build -t ${DOCKER_IMAGE}:latest .'
                }
            }
        }
        stage('Save and Transfer Docker Image') {
            steps {
                echo 'Saving and transferring Docker image to EC2...'
                sh """
                docker save ${DOCKER_IMAGE}:latest | gzip > YogaYoBack-0.0.1-SN
                """
```

```
            sshPublisher(publishers: [
                sshPublisherDesc(
                    configName: 'EC2-SERVER',
                    transfers: [
                        sshTransfer(
                            sourceFiles: 'YogaYoBack-0.0.1-SNAPSHOT.tar.gz'
                        )
                    ]
                )
            ])
        }
    }
    stage('Deploy to EC2') {
        steps {
            echo 'Deploying the application on EC2...'
            sshPublisher(publishers: [
                    sshPublisherDesc(
                        configName: 'EC2-SERVER',
                        transfers: [
                            sshTransfer(
                                execCommand: """
                                    mkdir -p ${DOCKER_PATH}
                                    docker stop ${DOCKER_CONTAINER} || true
                                    docker rm ${DOCKER_CONTAINER} || true
                                    docker rmi ${DOCKER_IMAGE}:latest || true
                                    docker load < ${DOCKER_PATH}/YogaYoBack-0.0.1-

                                    docker run -d --name ${DOCKER_CONTAINER} \
                                        --network ubuntu_app-network \
                                        -e SPRING_PROFILES_ACTIVE='prod' \
                                        -p ${DOCKER_PORT}:${DOCKER_PORT} \
                                        -e SERVER_PORT=${DOCKER_PORT} \
                                        -e DB_PROD_URL=${DB_PROD_URL} \
                                        -e DB_PROD_USERNAME=${DB_PROD_USERNAM
                                        -e DB_PROD_PASSWORD=${DB_PROD_PASSWOR
                                        -e your-bucket-name=${S3_BUCKET_NAME} \
                                        -e yourAccessKey=${S3_ACCESS_KEY} \
                                        -e yourSecretKey=${S3_SECRET_KEY} \
```

```
                                -e JWT-SECRET=${JWT_SECRET_KEY} \
                                -e TZ=Asia/Seoul \
                                ${DOCKER_IMAGE}:latest
                            """.stripIndent()
                        )
                    ]
                )
            ])
        }
    }
}
post {
    success {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdou
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnS
            mattermostSend (color: 'good',
            message: "빌드 성공!: ${env.JOB_NAME} #${env.BUILD_NUMBER}
            endpoint: 'https://meeting.ssafy.com/hooks/fuwwodco37nb9jmxh
            channel: 'd104jenkins'
            )
        }
    }
    failure {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdou
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnS
            mattermostSend (color: 'danger',
            message: "빌드 실패ㅠ: ${env.JOB_NAME} #${env.BUILD_NUMBER
            endpoint: 'https://meeting.ssafy.com/hooks/fuwwodco37nb9jmxh
            channel: 'd104jenkins'
            )
        }
    }
}
```

3.4. Nginx.conf 파일

```
server {
  listen 80; # 80포트로 받을 때
  server_name j12d104.p.ssafy.io; #도메인주소, 없을경우 localhost
  location ^~ /.well-known/acme-challenge/ {
        root /var/www/html;
   }
  location / {
        return 301 https://$host$request_uri;
   }
}
server{
  listen 443 ssl http2;
  server_name j12d104.p.ssafy.io;

  # ssl 인증서 적용하기
  ssl_certificate /etc/letsencrypt/live/j12d104.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/j12d104.p.ssafy.io/privkey.pem;

  proxy_set_header Host $http_host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header X-Forwarded-Proto $scheme;

   # 기본 CORS 설정
  add_header 'Access-Control-Allow-Origin' '*' always;
  add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, PAT
  add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested
  add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Rar

  location / { # location 이후 특정 url을 처리하는 방법을 정의(여기서는 / -> 즉, 모든 r
    # 프리플라이트 요청 처리
    if ($request_method = 'OPTIONS') {
      add_header 'Access-Control-Allow-Origin' '*';
      add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, P
      add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Request
      add_header 'Access-Control-Max-Age' 1728000;
      add_header 'Content-Type' 'text/plain; charset=utf-8';
      add_header 'Content-Length' 0;
```

```
      return 204;
    }
    proxy_pass http://43.203.169.28:8080; # Request에 대해 어디로 리다이렉트하는
  }

  location /jenkins/{
    proxy_pass http://43.203.169.28:18080; # Request에 대해 어디로 리다이렉트하
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";  # WebSocket 지원
  }

  location /ws/ {
   proxy_pass http://43.203.169.28:8080/ws;
   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;
   proxy_read_timeout 3600;
  }

}
```

3.5. 환경 변수 설정 (.env 파일)

프로젝트 루트에 `.env` 파일을 생성하고 다음과 같이 설정합니다.

```
DB_URL=MYSQL_DB_URL_LOCAL
DB_USERNAME=MYSQL_DB_USERNAME_LOCAL
DB_PASSWORD=MYSQL_DB_PASSWORD_LOCAL

DB_PROD_URL=MYSQL_DB_PASSWORD_PROD
DB_PROD_USERNAME=MYSQL_DB_PASSWORD_PROD
DB_PROD_PASSWORD=MYSQL_DB_PASSWORD_PROD

your-bucket-name=AWS_S3_BUCKET_NAME
yourAccessKey=AWS_S3_ACCESS_KEY
```

```
yourSecretKey=AWS_S3_SECRET_KEY

JWT-SECRET=JWT_SECRET_KEY
```

⚠️ **보안 주의사항**

- `.env` 파일을 **Git에 커밋하면 안 됩니다**. `.gitignore`에 추가하세요.
- EC2에서는 `/etc/environment`에 환경 변수를 추가할 수도 있습니다.

4. 외부 서비스 설정

4.1 AWS S3 설정

4.1.1 AWS S3 버킷 생성

- AWS 콘솔에서 `yogayo` 버킷 생성
- 공개 액세스 차단 유지
- IAM에서 S3 액세스 권한 부여

5. 서버 배포 (EC2)

5.1. EC2에 파일 전송

```
scp -i your-key.pem ${DockerFile 경로} ubuntu@your-ec2-ip:/home/ubuntu/   #도커파일이동
scp -i your-key.pem ${DockerCompose.yml 경로} ubuntu@your-ec2-ip:/home/ubuntu/ #도커 컴포즈 파일 이동
scp -i your-key.pem ${.env 경로} ubuntu@your-ec2-ip:/home/ubuntu/ #env 파일 이동
```

5.2. 도커 컴포즈 파일 실행

```
docker-compose up
```

`docker-compose up`을 실행시켜 `SpringBoot`, `MySQL`, `Jenkins` 컨테이너를 실행합니다

6. EC2 포트매핑

Port	프로그램
80	HTTP
443	HTTPS
8080	Spring Boot
18080	Jenkins
3306	MySQL

7. 배포흐름

- `docker-compose up` 명령어를 실행하여 **Spring Boot**, **MySQL**, **Jenkins** 컨테이너를 실행합니다.
- 개발자는 `develop-back` 브랜치에 코드를 **푸시**합니다.
- **GitLab Webhook**이 푸시 이벤트를 감지하여 **Jenkins**에 전달합니다.
- Jenkins는 해당 이벤트를 기반으로 **Spring Boot 프로젝트를 빌드**합니다.
- 빌드된 `.jar` 파일을 사용하여 **새로운 Spring Boot 컨테이너**를 실행합니다.