포팅 메뉴얼

기술 스택

- **Cooperation**

`gitlab` `notion` `jira` `mattermost`

- **Mobile**

`DART SDK:3.3.4` `flutter:3.19.5`

- **Backend**

`java:17.0.9` `spring boot:3.2.5`

`Python:3.12.1` `flask:3.03`

`gradle: 8.4` `JPA`

`quartz:2.3.2` `openai-gpt3-java:api:0.18.2` `Hibernate:6.4.4.Final`

- **DB**

`MYSQL 8.3.0` `SQLite:3.45.3`

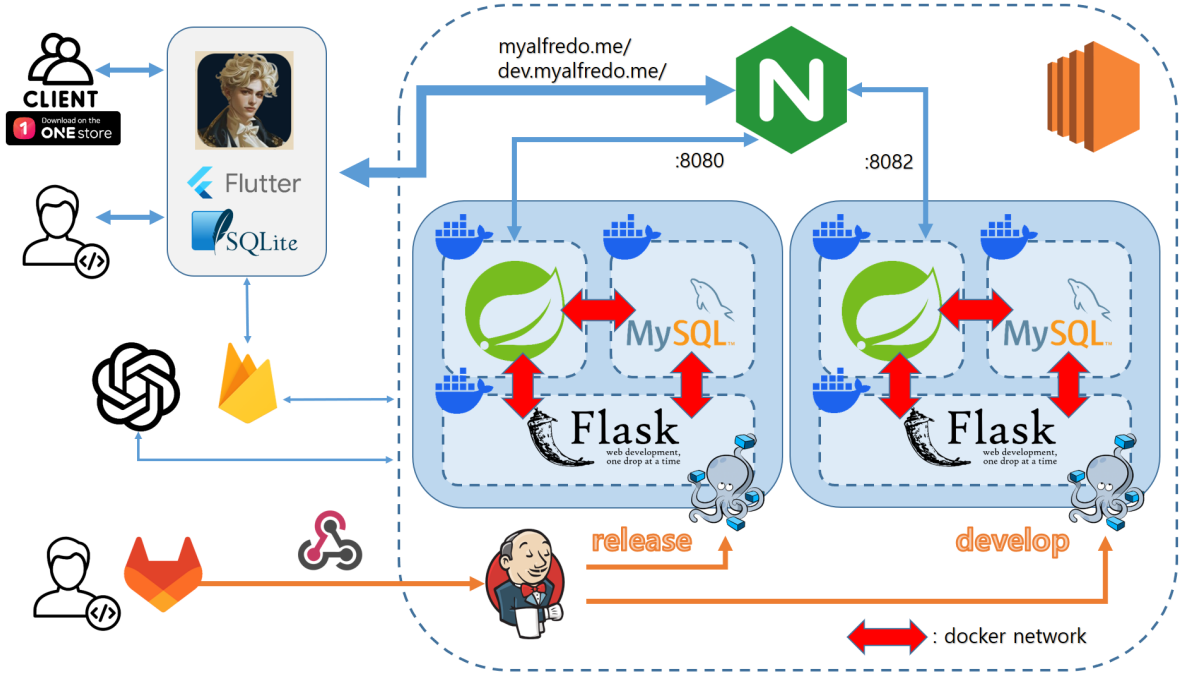
- **Cloud**

`firebase-admin:9.2.0`

- **Infra**

`ubuntu:20.04.6 LTS` `Jenkins 2.456` `nginx:1.18.0` `Docker 26.1.1`

인프라 아키텍처



설치 상세

- EC2 접속

```
ssh -i {pem_key}.pem ubuntu@{server_domain_name}
```

- 인스턴스 패키지 업데이트

```
sudo apt-get update
sudo apt-get upgrade
```

- TimeZone 설정

```
sudo timedatectl set-timezone Asia/Seoul
```

- 방화벽 설정

```
sudo ufw allow 80
sudo ufw allow 8080/tcp # 배포서버
sudo ufw allow 8082/tcp # 개발서버
```

- java 설치

```
sudo apt install openjdk-17-jdk
```

- NGINX 설치

```
sudo apt install nginx
sudo systemctl status nginx
```

- 도커 설치

```
# 충돌 방지 위한 기존 도커 이미지 전부 삭제
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg;

sudo apt-get update

#패키지 설치
sudo apt-get install ca-certificates curl

#키 추가
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# apt 소스에 추가
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# 버전 체크
docker --version

#사용자 확인
whoami

#테스트
docker run hello-world
docker: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Head "http://%
See 'docker run --help'.

# docker group 생성
sudo groupadd docker # groupadd: group 'docker' already exists

# 그룹 가입
sudo usermod -aG docker ubuntu
newgrp docker
```

```
sudo service docker restart

# 재 테스트
docker run hello-world

# 테스트 확인 후 이미지 삭제
docker stop fc79d71b0a61
docker rm fc79d71b0a61 # 반드시 컨테이너 ID로 삭제
docker rmi d2c94e258dcb # 이미지 ID로 삭제
```

- Jenkins 설치

```
# 키 추가
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

# 저장소 추가
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

# 설치
sudo apt-get update
sudo apt-get install jenkins

#포트 번호 변경
sudo vi /lib/systemd/system/jenkins.service # Environment="JENKINS_PORT=8081" 로 수정

# ufw 허용 추가
sudo ufw allow 8081
sudo ufw status

# jenkins 재시작
sudo systemctl restart jenkins
```

- Jenkins 계정 설정

```
# admin unlock
sudo cat /var/lib/jenkins/secrets/initialAdminPassword #발급되는 패스워드 저장

# 추천 설치 실패 # 플러그인 설치 경로 변경 필요
# 계정, pw, 이름 등록
# 설정 - 플러그인 - 플러그인 주소 변경
# jenkins 재시작
sudo systemctl restart jenkins

# 원활한 다운로드를 위한 jenkins 업데이트 센터 변경
cd /var/lib/jenkins
sudo mkdir update-center-rootCAs
sudo wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt -O ./update-center-rootCAs/updat
sudo chown -R jenkins:jenkins update-center-rootCAs

sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com/lework/jenkins-update-center/

sudo systemctl restart jenkins
```

- certbot(Let's encrypt) 을 통한 SSL 인증서 발급

```
sudo certbot --nginx -d myalfredo.me # 배포 도메인
sudo certbot --nginx -d dev.myalfredo.me # 개발 도메인

# 테스트
sudo certbot renew --dry-run
```

- /etc/nginx/sites-available/default 설정

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;
        server_name myalfredo.me; # managed by Certbot

        location / {
                return 301 https://$host$request_uri;
        }

}

server {
        listen 443 ssl;
        listen [::]:443 ssl ipv6only=on;
        server_name myalfredo.me;

        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;

        ssl_certificate /etc/letsencrypt/live/myalfredo.me/fullchain.pem; # managed by Certbot
        ssl_certificate_key /etc/letsencrypt/live/myalfredo.me/privkey.pem; # managed by Certbot
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

        charset utf-8;

        location /api/ {
                proxy_pass http://localhost:8080;
                proxy_set_header Host $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;
        }
        location / {
            try_files $uri $uri/ =404;
        }

}

server {
    if ($host = dev.myalfredo.me) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80;
        listen [::]:80;
        server_name dev.myalfredo.me;

        location / {
                return 301 https://$host$request_uri;
        }


}

server {
        listen 443 ssl;
        listen [::]:443 ssl;
        server_name dev.myalfredo.me;

        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;

        ssl_certificate /etc/letsencrypt/live/dev.myalfredo.me/fullchain.pem; # managed by Certbot
        ssl_certificate_key /etc/letsencrypt/live/dev.myalfredo.me/privkey.pem; # managed by Certbot
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

        charset utf-8;

        location /api/ {
                proxy_pass http://localhost:8082;
                proxy_set_header Host $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;
```

```
        }
        location / {
                try_files $uri $uri/ =404;
        }
}
```

- 도커 네트워크 생성

```
docker network create my-alfredo # 배포 서버
docker network create my-alfredo-dev # 개발 서버
```

- 서버 설정 파일을 서버 내 디렉토리에서 관리

```
sudo mkdir -p /app/config/alfredo # 배포 서버 설정 파일
sudo mkdir -p /app/config/alfredo/dev # 개발 서버 설정 파일
```

Jenkins 설정

Plugin

```
Pipeline: REST APIVersion 2.34
Pipeline: Stage ViewVersion 2.34
Oracle Java SE Development Kit InstallerVersion 73.vddf737284550
Command Agent LauncherVersion 107.v773860566e2e
Authentication Tokens APIVersion 1.53.v1c90fd9191a_b_
JavadocVersion 243.vb_b_503b_b_45537
JavaScript GUI Lib: ACE Editor bundleVersion 1.1
Git serverVersion 117.veb_68868fa_027
Maven IntegrationVersion 3.23
JavaScript GUI Lib: Moment.js bundleVersion 1.1.1
Docker CommonsVersion 439.va_3cb_0a_6a_fb_29
JavaScript GUI Lib: Handlebars bundleVersion 3.0.8
Lockable ResourcesVersion 1255.vf48745da_35d0
Docker PipelineVersion 572.v950f58993843
Config File ProviderVersion 973.vb_a_80ecb_9a_4d0
Popper.js 2 APIVersion 2.11.6-4
Pipeline: Deprecated Groovy LibrariesVersion 612.v55f2f80781ef
HTML PublisherVersion 1.33
MapDB APIVersion 1.0.9-40.v58107308b_7a_7
Role-based Authorization StrategyVersion 727.vd344b_eec783d
Popper.js APIVersion 1.16.1-3
SubversionVersion 1256.vee91953217b_6
External Monitor Job TypeVersion 215.v2e88e894db_f8
Handy Uri Templates 2.x APIVersion 2.1.8-30.v7e777411b_148
jQueryVersion 1.12.4-1
WMI Windows AgentsVersion 1.8.1
Bootstrap 4 APIVersion 4.6.0-6
Pub-Sub "light" BusVersion 1.18
FavoriteVersion 2.208.v91d65b_7792a_c
JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI)Version 1.2.1
Common API for Blue OceanVersion 1.27.12
Bitbucket Branch SourceVersion 886.v44cf5e4ecec5
REST API for Blue OceanVersion 1.27.12
Parameterized TriggerVersion 806.vf6fff3e28c3e
Design LanguageVersion 1.27.12
Blue Ocean Core JSVersion 1.27.12
Web for Blue OceanVersion 1.27.12
JWT for Blue OceanVersion 1.27.12
Infrastructure plugin for Publish Over XVersion 0.22
Pipeline SCM API for Blue OceanVersion 1.27.12
REST Implementation for Blue OceanVersion 1.27.12
Run ConditionVersion 1.7
Pipeline implementation for Blue OceanVersion 1.27.12
Server Sent Events (SSE) GatewayVersion 1.26
DataTables.net APIVersion 2.0.6-1
Git Pipeline for Blue OceanVersion 1.27.12
Dashboard for Blue OceanVersion 1.27.12
```

```
NodeJSVersion 1.6.1
GitHub Pipeline for Blue OceanVersion 1.27.12
```

GitLab 설정

- Jenkins 관리 - System - GitLab - GitLab Host URL 등록
- GitLab의 프로젝트 페이지에서 신규 access-token 발급 - scopes 권한 부여
- Jenkins 관리 - Credentials에 access-token 등록

Item 생성 및 WebHook 연결 - alfredoBuild, alfredoDev

- Pipeline 지정
- GitLab Connection 설정
- Build Triggers - Build when a change is pushed to GitLab. GitLab webhook URL 설정
  - Comment (regex) for triggering a build - Secret token Gererate 값 복사
- GitLab의 프로젝트 페이지 Setting
  - Webhooks - URL에 Jenkins item 등록
  - Secret Token에 복사한 값 등록
  - Trigger - Push Event - Regular expression - `^{branch}/`
  - 배포 서버 Item alfredoBuild는 `release` , 개발 서버 Item alfredoDev는 `BE/develop` 입력 후 save
- Pipeline - Pipeline script `release`

```
pipeline {
    agent any

    environment {
        DOCKER_COMPOSE_FILE = '/app/config/alfredo/docker-compose.yml'
        SPRING_SERVICE = 'spring'
        MYSQL_SERVICE = 'database'
        FLASK_SERVICE = 'flask'

        PROJECT_PATH = '/var/lib/jenkins/workspace/alfredoBuild/BE/alfredo'
        FLASK_PATH = '/var/lib/jenkins/workspace/alfredoBuild/flask_alfredo/flask_alfredo'
        CONFIG_PATH = '/app/config/alfredo'
    }

    stages {
        stage('Clone repository') {
            steps {
                echo 'git clone start'
                git(
                    branch: 'release',
                    credentialsId: 'gitlab-auth',
                    url: '{gitlab_project_url}'
                )
                echo 'git clone complete'
            }
        }

        stage('Copy Flask Environment Files') {
            steps {
                script {
                    echo "Copy .env file and Dockerfile for Flask."
                    sh "cp ${CONFIG_PATH}/.env ${FLASK_PATH}/.env"
                    sh "cp ${CONFIG_PATH}/Dockerfile.flask ${FLASK_PATH}/Dockerfile.flask"
                }
            }
        }

        stage('Prepare Spring Container') {
            steps {
                script {
                    def springRunning = sh(script: "docker ps -q -f name=${SPRING_SERVICE}", returnStdout: true).trim()
                    if (springRunning) {
                         echo "${SPRING_SERVICE} container is running. Stopping and removing..."
                        sh "docker compose -f ${DOCKER_COMPOSE_FILE} stop ${SPRING_SERVICE}"
                        sh "docker compose -f ${DOCKER_COMPOSE_FILE} rm -f ${SPRING_SERVICE}"

                    } else {
                        echo "${SPRING_SERVICE} container is not running."
                    }
```

```
                def imageExists = sh(script: "docker images -q alfredo-spring:latest", returnStdout: true).trim()
                if (imageExists) {
                    echo "Image alfredo-spring:latest exists. Removing..."
                    sh "docker rmi alfredo-spring:latest"
                } else {
                    echo "Image alfredo-spring:latest does not exist."
                }
            }
        }
    }

    stage('Check and Prepare MySQL Container') {
        steps {
            script {
                def dbContainerId = sh(script: "docker compose -f ${DOCKER_COMPOSE_FILE} ps -q ${MYSQL_SERVICE}", return
                if (dbContainerId) {
                    echo "${MYSQL_SERVICE} container is running."
                } else {
                    echo "${MYSQL_SERVICE} container is not running. Attempting to start..."
                    sh "docker compose -f ${DOCKER_COMPOSE_FILE} up -d ${MYSQL_SERVICE}"
                }
            }
        }
    }

    stage('Set Spring Container Environment') {
        steps {
            script {
                echo "copy environment files."
                sh "mkdir -p ${PROJECT_PATH}/src/main/resources"
                sh "cp ${CONFIG_PATH}/application.properties ${PROJECT_PATH}/src/main/resources/"
                sh "cp ${CONFIG_PATH}/alfredo104-firebase-adminsdk-{user_id}.json ${PROJECT_PATH}/"
                sh "cp ${CONFIG_PATH}/Dockerfile ${PROJECT_PATH}/"
                sh "cp ${CONFIG_PATH}/gradle-wrapper.properties ${PROJECT_PATH}/gradle/wrapper/"

                echo "Checking firebase.sdk path from application.properties:"
                def firebaseSdkPath = sh(script: "cat ${PROJECT_PATH}/src/main/resources/application.properties | grep f
                echo "firebase.sdk path is: ${firebaseSdkPath}"
            }
        }
    }

    stage('Prepare Flask Container') {
        steps {
            script {
                def flaskRunning = sh(script: "docker ps -q -f name=${FLASK_SERVICE}", returnStdout: true).trim()
                if (flaskRunning) {
                    echo "${FLASK_SERVICE} container is running. Stopping and removing..."
                    sh "docker compose -f ${DOCKER_COMPOSE_FILE} stop ${FLASK_SERVICE}"
                    sh "docker compose -f ${DOCKER_COMPOSE_FILE} rm -f ${FLASK_SERVICE}"
                } else {
                    echo "${FLASK_SERVICE} container is not running."
                }

                def imageExists = sh(script: "docker images -q alfredo-flask:latest", returnStdout: true).trim()
                if (imageExists) {
                    echo "Image alfredo-flask:latest exists. Removing..."
                    def containers = sh(script: "docker ps -a -q --filter ancestor=alfredo-flask:latest", returnStdout:
                    if (containers) {
                        sh "docker rm -f ${containers}"
                    }
                    sh "docker rmi -f alfredo-flask:latest"
                } else {
                    echo "Image alfredo-flask:latest does not exist."
                }
            }
        }
    }

    stage('Build and Run Spring Container') {
        steps {
            script {
                echo "Build Spring Container."
                sh "chmod +x ${PROJECT_PATH}/gradlew"
                // sh "cd ${PROJECT_PATH} && ./gradlew clean build --refresh-dependencies"
                sh "cd ${PROJECT_PATH} && ./gradlew clean build -x test"
```

```
                    echo "Running Spring Container using Docker Compose."
                    sh "cd ${CONFIG_PATH}"
                    sh "docker compose -f ${DOCKER_COMPOSE_FILE} up --build -d ${SPRING_SERVICE}"
                }
            }
        }

    }

    post {
        always {
            echo "Build and deployment process completed."
        }
        success {
            mattermostSend color: 'good', message: 'Service Server Build, Test and Deployment Successful', channel: 'B104-GI
        }
        failure {
            mattermostSend color: 'danger', message: 'Service Server Build, Test or Deployment Failed', channel: 'B104-GIT',
        }
        unstable {
            mattermostSend color: 'warning', message: 'Service Server Build Unstable', channel: 'B104-GIT', webhookUrl: 'htt
        }
        aborted {
            mattermostSend color: 'warning', message: 'Service Server Build Aborted', channel: 'B104-GIT', webhookUrl: 'http
        }
    }
}
```

- Pipeline script `BE/develop`

```
pipeline {
    agent any

    environment {
        DOCKER_COMPOSE_FILE = '/app/config/alfredo/dev/docker-compose.yml'
        SPRING_SERVICE = 'spring-dev'
        MYSQL_SERVICE = 'database-dev'
        FLASK_SERVICE = 'flask-dev'

        PROJECT_PATH = '/var/lib/jenkins/workspace/alfredoDev/BE/alfredo'
        FLASK_PATH = '/var/lib/jenkins/workspace/alfredoDev/flask_alfredo/flask_alfredo'
        CONFIG_PATH = '/app/config/alfredo/dev'
    }

    stages {
        stage('Clone repository') {
            steps {
                echo 'git clone start'
                git(
                    branch: 'BE/develop',
                    credentialsId: 'gitlab-auth',
                    url: '{gitlab_project_url}'
                )
                echo 'git clone complete'
            }
        }

        stage('Copy Flask Environment Files') {
            steps {
                script {
                    echo "Copy .env file and Dockerfile for Flask."
                    sh "cp ${CONFIG_PATH}/.env ${FLASK_PATH}/.env"
                    sh "cp ${CONFIG_PATH}/Dockerfile.flask ${FLASK_PATH}/Dockerfile.flask"
                }
            }
        }

        stage('Prepare Spring Container') {
            steps {
                script {
                    def springRunning = sh(script: "docker ps -q -f name=${SPRING_SERVICE}", returnStdout: true).trim()
                    if (springRunning) {
                        sh "docker compose -f ${DOCKER_COMPOSE_FILE} stop ${SPRING_SERVICE}"
                        sh "docker compose -f ${DOCKER_COMPOSE_FILE} rm -f ${SPRING_SERVICE}"
```

```
                    sh "docker rmi alfredo-spring-dev:latest"
                }
            }
        }
    }

    stage('Check and Prepare MySQL Container') {
        steps {
            script {
                def dbContainerId = sh(script: "docker compose -f ${DOCKER_COMPOSE_FILE} ps -q ${MYSQL_SERVICE}", return
                if (dbContainerId) {
                    echo "${MYSQL_SERVICE} container is running."
                } else {
                    echo "${MYSQL_SERVICE} container is not running. Attempting to start..."
                    sh "docker compose -f ${DOCKER_COMPOSE_FILE} up -d ${MYSQL_SERVICE}"
                }
            }
        }
    }

    stage('Set Spring Container Environment') {
        steps {
            script {
                echo "copy environment files."
                sh "mkdir -p ${PROJECT_PATH}/src/main/resources"
                sh "cp ${CONFIG_PATH}/application.properties ${PROJECT_PATH}/src/main/resources/"
                sh "cp ${CONFIG_PATH}/alfredo104-firebase-adminsdk-{user_id}.json ${PROJECT_PATH}/"
                sh "cp ${CONFIG_PATH}/Dockerfile ${PROJECT_PATH}/"
                sh "cp ${CONFIG_PATH}/gradle-wrapper.properties ${PROJECT_PATH}/gradle/wrapper/"

                echo "Checking firebase.sdk path from application.properties:"
                def firebaseSdkPath = sh(script: "cat ${PROJECT_PATH}/src/main/resources/application.properties | grep f
                echo "firebase.sdk path is: ${firebaseSdkPath}"
            }
        }
    }

    stage('Prepare Flask Container') {
        steps {
            script {
                def flaskRunning = sh(script: "docker ps -q -f name=${FLASK_SERVICE}", returnStdout: true).trim()
                if (flaskRunning) {
                    echo "${FLASK_SERVICE} container is running. Stopping and removing..."
                    sh "docker compose -f ${DOCKER_COMPOSE_FILE} stop ${FLASK_SERVICE}"
                    sh "docker compose -f ${DOCKER_COMPOSE_FILE} rm -f ${FLASK_SERVICE}"
                } else {
                    echo "${FLASK_SERVICE} container is not running."
                }

                def imageExists = sh(script: "docker images -q alfredo-flask-dev:latest", returnStdout: true).trim()
                if (imageExists) {
                    echo "Image alfredo-flask:latest exists. Removing..."
                    sh "docker rmi alfredo-flask-dev:latest"
                } else {
                    echo "Image alfredo-flask-dev:latest does not exist."
                }
            }
        }
    }

    stage('Build and Run Spring Container') {
        steps {
            script {
                echo "Build Spring Container."
                sh "chmod +x ${PROJECT_PATH}/gradlew"
                // sh "cd ${PROJECT_PATH} && ./gradlew clean build --refresh-dependencies"
                sh "cd ${PROJECT_PATH} && ./gradlew clean build -x test"

                echo "Running Spring Container using Docker Compose."
                sh "cd ${CONFIG_PATH}"
                sh "docker compose -f ${DOCKER_COMPOSE_FILE} up --build -d ${SPRING_SERVICE}"
            }
        }
    }

}
```

```
    post {
        always {
            echo "Build and deployment process completed."
        }
        success {
            mattermostSend color: 'good', message: 'Develop Server Build, Test and Deployment Successful', channel: 'B104-GI
        }
        failure {
            mattermostSend color: 'danger', message: 'Develop Server Build, Test or Deployment Failed', channel: 'B104-GIT',
        }
        unstable {
            mattermostSend color: 'warning', message: 'Develop Server Build Unstable', channel: 'B104-GIT', webhookUrl: 'htt
        }
        aborted {
            mattermostSend color: 'warning', message: 'Develop Server Build Aborted', channel: 'B104-GIT', webhookUrl: 'http
        }
    }

}
```

서버 디렉토리 내 설정 파일

- `/app/config/alfredo` 배포 서버 설정 파일 디렉토리
  - Dockerfile

```
FROM openjdk:17
WORKDIR /server
COPY ./build/libs/alfredo-0.0.1-SNAPSHOT.jar alfredo.jar
ENTRYPOINT ["java", "-jar", "alfredo.jar"]
```

  - Dockerfile.flask

```
# 베이스 이미지 설정
FROM python:3.12.3

# 작업 디렉토리 설정
WORKDIR /app

# 필요한 패키지 설치
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Flask 앱 파일 복사
COPY . /app

# 환경 변수 로드
ENV FLASK_APP=app.py

# Flask 서버 실행
CMD ["flask", "run", "--host=0.0.0.0"]
```

  - docker-compose.yml

```
services:
  spring:
    build:
      context: /var/lib/jenkins/workspace/alfredoBuild/BE/alfredo
      dockerfile: Dockerfile
    image: alfredo-spring:latest
    container_name: alfredo-spring
    restart: always
    depends_on:
      - database
      - flask
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://database:3306/alfredo
      SPRING_DATASOURCE_USERNAME: {user_name}
      SPRING_DATASOURCE_PASSWORD: {user_password}
      TZ: 'Asia/Seoul'
      FIREBASE_SDK_PATH: /app/alfredo104-firebase-adminsdk-{user_id}.json
    volumes:
```

```
      - /var/lib/jenkins/workspace/alfredoBuild/BE/alfredo/alfredo104-firebase-adminsdk-{user_id}.json:/app/alfredo104-fi
    networks:
      - my-alfredo

  flask:
    build:
      context: /var/lib/jenkins/workspace/alfredoBuild/flask_alfredo/flask_alfredo
      dockerfile: Dockerfile.flask
    image: alfredo-flask:latest
    container_name: alfredo-flask
    restart: always
    depends_on:
      - database
    env_file: /var/lib/jenkins/workspace/alfredoBuild/flask_alfredo/flask_alfredo/.env
    environment:
      TZ: 'Asia/Seoul'
    networks:
      - my-alfredo

  database:
    image: mysql:latest
    container_name: alfredo-database
    environment:
      MYSQL_ROOT_PASSWORD: {root_password}
      MYSQL_DATABASE: {db_name}
      MYSQL_USER: {user_name}
      MYSQL_PASSWORD: {user_password}
      MYSQL_CHARACTER_SET_SERVER: utf8mb4
      MYSQL_COLLATION_SERVER: utf8mb4_unicode_ci
      TZ: 'Asia/Seoul'
    volumes:
      - alfredo-db:/var/lib/mysql
      - /app/config/alfredo/init.sql:/docker-entrypoint-initdb.d/init.sql
    networks:
      - my-alfredo

networks:
  my-alfredo:
    external: true

volumes:
  alfredo-db:
```

- alfredo104-firebase-adminsdk-{user_id}.json

```
{
  "type": "service_account",
  "project_id": "{project_id}",
  "private_key_id": "{key_id}",
  "private_key": "{key}",
  "client_email": "{client_email}",
  "client_id": "{client_id}",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "{client_x509_cert_url}",
  "universe_domain": "googleapis.com"
}
```

- gradle-wrapper.properties

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-8.4-bin.zip
networkTimeout=10000
validateDistributionUrl=true
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

- application.properties

```
spring.application.name=alfredo

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://database:3306/alfredo?serverTimezone=Asia/Seoul&characterEncoding=UTF-8&useSSL=false
spring.datasource.username={user_name}
spring.datasource.password={user_password}
```

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

spring.jpa.hibernate.ddl-auto=validate

firebase.sdk=${FIREBASE_SDK_PATH}
firebase.api.url=https://fcm.googleapis.com/v1/projects/alfredo104/messages:send

openai.api.key={openai_api_key}
openai.api.url=https://api.openai.com/v1/audio/speech

flask.url=http://alfredo-flask:5000
```

- init.sql

```
-- create database
CREATE DATABASE IF NOT EXISTS alfredo;

-- add new user
CREATE USER '{user_name}'@'%' IDENTIFIED BY '{user_password}';

--grant permission to user
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP ON alfredo.* TO '{user_name}'@'%';

--apply changes
FLUSH PRIVILEGES;
```

- `/app/config/alfred/dev` 개발 서버 설정 파일 디렉토리
  - Dockerfile

```
FROM openjdk:17
WORKDIR /server
COPY ./build/libs/alfredo-0.0.1-SNAPSHOT.jar alfredo.jar
ENTRYPOINT ["java", "-jar", "alfredo.jar"]
```

  - Dockerfile.flask

```
# 베이스 이미지 설정
FROM python:3.12.3

# 작업 디렉토리 설정
WORKDIR /app

# 필요한 패키지 설치
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Flask 앱 파일 복사
COPY . /app

# 환경 변수 로드
ENV FLASK_APP=app.py

# Flask 서버 실행
CMD ["flask", "run", "--host=0.0.0.0"]
```

  - docker-compose.yml

```
services:
  spring-dev:
    build:
      context: /var/lib/jenkins/workspace/alfredoDev/BE/alfredo
      dockerfile: Dockerfile
    image: alfredo-spring-dev:latest
    container_name: alfredo-spring-dev
    restart: always
    depends_on:
      - database-dev
      - flask-dev
    ports:
      - "8082:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://database-dev:3306/alfredo
      SPRING_DATASOURCE_USERNAME: {user_name}
      SPRING_DATASOURCE_PASSWORD: {user_password}
      TZ: 'Asia/Seoul'
```

```
      FIREBASE_SDK_PATH: /app/alfredo104-firebase-adminsdk-{user_id}.json
    volumes:
      - /var/lib/jenkins/workspace/alfredoDev/BE/alfredo/alfredo104-firebase-adminsdk-{user_id}.json:/app/alfredo104-fire
    networks:
      - my-alfredo-dev

  flask-dev:
    build:
      context: /var/lib/jenkins/workspace/alfredoDev/flask_alfredo/flask_alfredo
      dockerfile: Dockerfile.flask
    image: alfredo-flask:latest
    container_name: alfredo-flask-dev
    restart: always
    depends_on:
      - database-dev
    env_file: /var/lib/jenkins/workspace/alfredoDev/flask_alfredo/flask_alfredo/.env
    environment:
      TZ: 'Asia/Seoul'
    networks:
      - my-alfredo-dev

  database-dev:
    image: mysql:latest
    container_name: alfredo-database-dev
    environment:
      MYSQL_ROOT_PASSWORD: {root_password}
      MYSQL_DATABASE: {db_name}
      MYSQL_USER: {user_name}
      MYSQL_PASSWORD: {user_password}
      MYSQL_CHARACTER_SET_SERVER: utf8mb4
      MYSQL_COLLATION_SERVER: utf8mb4_unicode_ci
      TZ: 'Asia/Seoul'
    volumes:
      - alfredo-db-dev:/var/lib/mysql
      - /app/config/alfredo/dev/init.sql:/docker-entrypoint-initdb.d/init.sql
    networks:
      - my-alfredo-dev

networks:
  my-alfredo-dev:
    external: true

volumes:
  alfredo-db-dev:
```

- alfredo104-firebase-adminsdk-{user_id}.json

```
{
  "type": "service_account",
  "project_id": "{project_id}",
  "private_key_id": "{key_id}",
  "private_key": "{key}",
  "client_email": "{client_email}",
  "client_id": "{client_id}",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "{client_x509_cert_url}",
  "universe_domain": "googleapis.com"
}
```

- gradle-wrapper.properties

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-8.4-bin.zip
networkTimeout=10000
validateDistributionUrl=true
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

- application.properties

```
spring.application.name=alfredo

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://database-dev:3306/alfredo?serverTimezone=Asia/Seoul&characterEncoding=UTF-8&useSSL=fal
```

```
spring.datasource.username={user_name}
spring.datasource.password={user_password}
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

spring.jpa.hibernate.ddl-auto=update

firebase.sdk=${FIREBASE_SDK_PATH}
firebase.api.url=https://fcm.googleapis.com/v1/projects/alfredo104/messages:send

openai.api.key={openai_api_key}
openai.api.url=https://api.openai.com/v1/audio/speech

flask.url=http://alfredo-flask-dev:5000
```

- init.sql

```
-- create database
CREATE DATABASE IF NOT EXISTS alfredo;

-- add new user
CREATE USER '{user_name}'@'%' IDENTIFIED BY '{user_password}';

--grant permission to user
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP ON alfredo.* TO '{user_name}'@'%';

--apply changes
FLUSH PRIVILEGES;
```

외부 서비스 정보

- 소셜 로그인
  - Google Firebase
- 외부 API
  - OPENAI TTS

덤프 파일

```
INSERT INTO basic_routine (start_time, alarm_sound, basic_routine_title, memo) VALUES
('06:00', 'Morning Glory', '기상', NULL),
('06:30', 'Morning Glory', '명상', NULL),
('07:00', 'Morning Glory', '모닝커피', NULL),
('14:00', 'Morning Glory', '공부', NULL),
('19:00', 'Morning Glory', '운동', NULL),
('20:00', 'Morning Glory', '독서', NULL),
('21:00', 'Morning Glory', '샤워', NULL),
('22:00', 'Morning Glory', '스트레칭', NULL),
('23:00', 'Morning Glory', '취침', NULL);

INSERT INTO basic_routine_days (basic_routine_id, days) VALUES
(1, 'MON'), (1, 'TUE'), (1, 'WED'), (1, 'THU'), (1, 'FRI'),
(2, 'MON'), (2, 'TUE'), (2, 'WED'), (2, 'THU'), (2, 'FRI'),
(3, 'MON'), (3, 'TUE'), (3, 'WED'), (3, 'THU'), (3, 'FRI'),
(4, 'MON'), (4, 'TUE'), (4, 'WED'), (4, 'THU'), (4, 'FRI'),
(5, 'MON'), (5, 'TUE'), (5, 'WED'), (5, 'THU'), (5, 'FRI'),
(6, 'MON'), (6, 'TUE'), (6, 'WED'), (6, 'THU'), (6, 'FRI'),
(7, 'MON'), (7, 'TUE'), (7, 'WED'), (7, 'THU'), (7, 'FRI'),
(8, 'MON'), (8, 'TUE'), (8, 'WED'), (8, 'THU'), (8, 'FRI'),
(9, 'MON'), (9, 'TUE'), (9, 'WED'), (9, 'THU'), (9, 'FRI');
```

시연 시나리오

1. 설문조사 페이지에서 시작 - 설문 선택 - 시작하기

2. 루틴 페이지→ 추천받은 루틴 확인하고, 그 루틴중에 하나 현재시간에서 +1 알람 켜놓고 일정탭으로 가기,
3. 일정탭에서 일정 하나 만들때, 시작시간 종료시간 설정후 알람시간을 +2분뒤로 설정
4. 투두 가서 반복 할 일로 15일 이상 하나 설정하면 퀘스트 깨지는거 보여주기
5. 타이머 한번 보여주기, 체크 하나 하기
6. 캘린더 가서 먼스버전, 위크, 데이버전 보여주기
7. 마이페이지가서 생일이랑 ical 등록하고 투두 차트 반영된거 짧게 소개
8. 업적페이지 가서 업적 설명
9. 캘린더 가서 ical 등록된거 한 번 확인해주기
10. 메인페이지 가서 tts 실행
11. 상점가서 배경이랑 캐릭터 변경