포팅 메뉴얼 (사무국 제출)

1. Gitlab 소스 클론 이후 빌드 및 배포할 수 있도록 정리한 문서

각종 버전 및 명령어 정보

(1) 버전 관리 툴 정보

- git `2.30.2`
- 외부 서비스
    - Gitlab (SSAFY제공)
    - Webhook

(2) 빌드 관련 정보

- Open JDK `17`
    - Gradle `8.2.1`
- Node.js `18.18.0`
    - npm `9.8.1`
- Jenkins `2.422` (Docker)

1) [프론트] : Install & Build 정상 여부 확인

```
cd frontend/popplar/android/app
pwd
npm i
npm start
```

2) [백엔드] : Build 정상 여부 확인

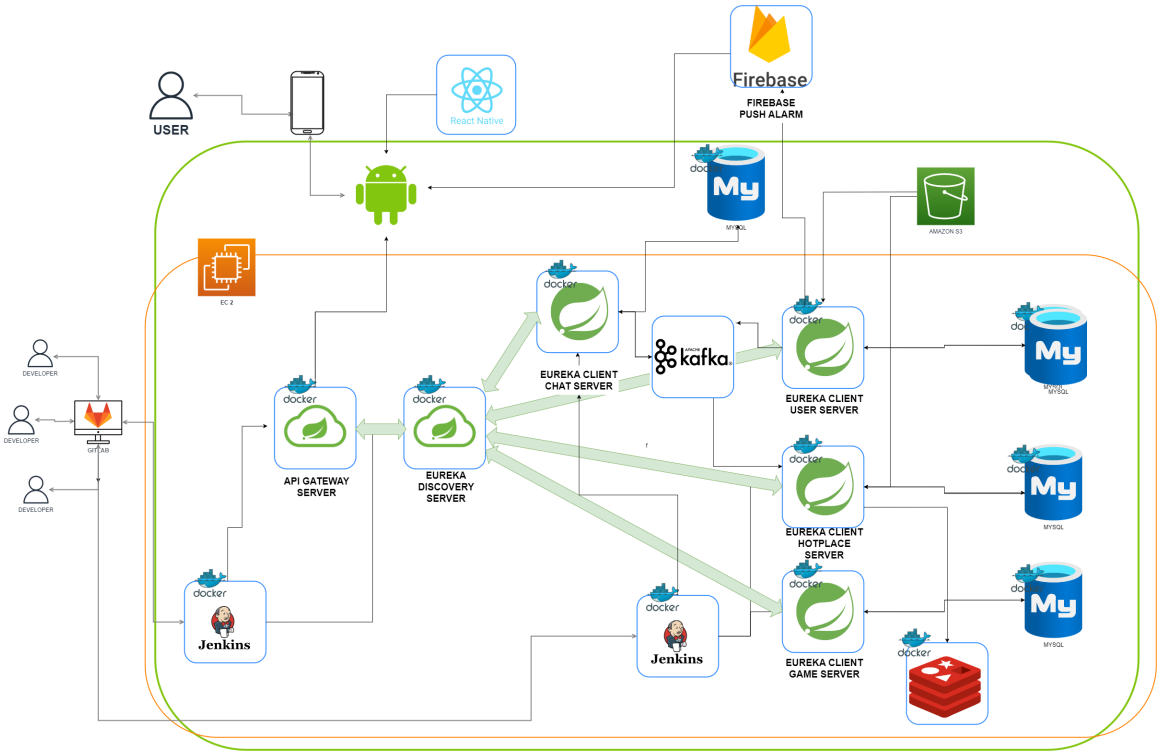
```
cd backend/{서비스명}
chmod +x gradlew
./gradlew build
```

2-2) 환경변수

```
#RDS
DB_URL=jdbc:mysql://newstocks-rds.czvfjrx99bcw.ap-northeast-2.rds.amazonaws.com:3306/hotspot_rds
DB_ID=joonsuk
DB_PWD=dhwnstjr12!

## OAuth
KAKAO_RESTAPI_KEY=0da056655f3ed1ec3ebd8325d19ac9f6
KAKAO_REDIRECT_URL=http://localhost:8201/member/login

GOOGLE_CLIENT_ID=109790637225-dl46o6lmeh4klauc53ji199aj4dnb8d4.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=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
GOOGLE_REDIRECT_URI=http://localhost:8080/member/login

SALT_A=5153
SALT_B=2477
SALT_C=6991

HOT_PLACE_URL=http://k9a705.p.ssafy.io:8200/hot-place
LIVE_CHAT_URL=http://k9a705.p.ssafy.io:8203/live-chat

SALT=PopplarKeyForEncryptionAndAuthenticationAndAuthorization

KAFKA_SERVER=localhost:9092
```

(3) 배포 관련 정보

- Docker 20.10.21
- 외부 서비스
  - AWS EC2 (SSAFY 제공)
  - AWS S3
  - Firebase Storage



1) [백엔드] : Dockerfile, Jenkinsfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8201
ENTRYPOINT ["java", "-jar", "app.jar"]
```

```
pipeline {
    agent any

    stages {
        stage('Gradle Build') {
            steps {
                dir('backend/member-service') {
                    // gradlew 실행 권한 부여
                    sh 'chmod +x gradlew'
                    // gradlew를 사용해 프로젝트를 빌드하며 테스트는 제외합니다.
                    sh './gradlew clean build -x test'
                }
            }
        }

        // docker-compose 기반 빌드
        stage('Docker Build') {
            steps {
                dir('backend/member-service') {
                    // 도커 컴포즈 빌드
                    echo "docker compose build"
                    sh "docker-compose -f docker-compose.yml build --no-cache"
```

```
// 일반 빌드가 deprecated 되어서, BuildKit을 사용하는 코드. 여기서는 안되서 이전 버젼으로 진행
//                     sh 'DOCKER_BUILDKIT=1 docker build -t herosof-trashbin:latest .'

                }
            }
        }

        stage('Deploy') {
            steps {
                // 도커 컴포즈 업
                dir('backend/member-service') {
                    sh "docker-compose -f docker-compose.yml up -d"
                }
                // 새로운 이미지로 'hot-place' 컨테이너를 백그라운드에서 실행
//                sh 'docker run -d --name hot-place -p 8200:8200 -u root popplar-hot-place:latest'
            }
        }

        // 완료 스테이지: 더이상 사용되지 않는 Docker 이미지를 제거합니다.
        stage('Finish') {
            steps {
                // 사용되지 않는 (dangling) 이미지를 찾아 제거합니다.
                sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
            }
        }
    }
}
```

3) application.yml

```
server:
  port: 8201



spring:
  #카프카 설정
  kafka:
    bootstrap-servers: ${KAFKA_SERVER}
    consumer:
      group-id: foo
      auto-offset-reset: earliest
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
    producer:
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
  application:
    name: member-service
  config:
    import: optional:file:.env[.properties]
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: ${DB_URL}
    username: ${DB_ID}
    password: ${DB_PWD}
    hikari:
```

```
      maximum-pool-size: 5
      minimum-idle: 5
      connection-timeout: 10000
      connection-init-sql: SELECT 1
      idle-timeout: 600000
      max-lifetime: 1800000
      auto-commit: true
  jpa:
    database: mysql
    database-platform: org.hibernate.dialect.MySQLDialect
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
        default_batch_fetch_size: 1000


eureka:
  instance:
    hostname: k9a705.p.ssafy.io
    instance-id: member-microservice-instance

  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://k9a705.p.ssafy.io:8761/eureka


logging:
  level:
    root: info
```

2. 프로젝트에서 사용하는 외부 서비스 정보를 정리한 문서

- 소셜 로그인
  - Google OAuth 2.0
  - Kakao Social Login
- AWS
  - AWS S3
  - AWS RDS

3. DB 덤프 파일 최신본

별첨

4. 시연 시나리오

<앱 시연>

1. **로그인하고 위치 정보 제공 동의**
2. **내가 원하는 공간에 사용자가 아지트를 생성할 수 있음**
3. **아지트 인근 반경 500미터 안의 사용자들은 해당 아지트에 입장 가능, 반경을 벗어나면 자동으로 아지트에서 나가지고 거리가 먼 참진 한의원의 경우 입장 불가하다.**
4. **QnA의 경우는 거리 밖에서 접근 가능,**
5. **아지트에 입장하면, 해당 아지트에 입장해있는 익명의 사용자들과 채팅 가능→채팅하는 거 보여주기(채팅 서비스 시연)**
6. **아지트에 입장하면, 주변에 아지트에 입장한 사용자들의 대략적인 위치 정보를 볼 수 있고 쪽지를 보내 소통할 수도 있음(아지트 비밀지도 서비스 시연)**
7. **아지트에 머무르면서 심심할 때에는 간단한 게임으로 아지트 사람들과 점수 경쟁을 통해 랭킹을 차지할 수 있고, 1등에게는 정복자라는 타이틀이 주어지며, 아지트에 정복자로 기록되며, 채팅에서 특수 효과를 받음(게임 시연)**
8. **내가 생성한 아지트에 방문하는 사용자들이 많아지면 아지트의 레벨이 올라가고, 다른 사람들이 생성한 아지트를 방문할 때마다 다양한 업적을 쌓을 수 있어 사용자에게 성취감을 제공함.(업적, 아지트 업그레이드 시연)**