

포팅 매뉴얼

목차

1. 사용 도구
2. 개발 환경
3. 외부 서비스
4. 환경 변수
5. 배포하기
6. 빌드 및 실행

1. 사용 도구

- 이슈 관리 : JIRA
- 형상 관리 : GitLab
- 커뮤니케이션 : Notion, Mattermost
- 디자인 : Figma
- CI/CD : Jenkins, Docker
- 실행 환경 : Android 14
- 배포 서버 : Amazon EC2

2. 개발 환경

Frontend(WEB)

name	version
VSCode	1.91.1
React	18.0.0

Next.js	14.2.15
TypeScript	5.6.3
Redux Toolkit	2.3.0
Tanstack-Query	5.59.16
Tailwind CSS	3.4.1

App(WEBVIEW)

VSCode	1.91.1
React	18.2.0
React Native	0.74.5
Android	14
Expo	51.0.28
TypeScript	5.6.3
Zustand	5.0.1
Styled-Components	6.1.13

Wear(Watch)

Android	13
Wear OS	4.0
Java	17

Arduino

Arduino	NANO 33 BLE
---------	-------------

Backend

name	version
Java	17
SpringBoot	3.3.5

Kafka	3.2.4
JPA	3.3.5
QueryDSL	5.0.0
rabbitmq	5.21.0
Spring Cloud Bus	4.1.2
spring Cloud Config	4.1.3
spring Cloud Gateway	4.1.5
spring Cloud Netflix Eureka	4.1.3
spring Cloud OpenFeign	4.1.3

3. 외부 서비스

coolsms

- 문자 서비스
- 개발/연동 → API Key 관리 → 새로운 API KEY 눌러서 키 생성

kakao

- 소셜 인증

4. 환경 변수 형태

프론트엔드 환경 변수 목록

- 구글 맵스 API 키 값
- 카카오 SDK 네이티브 키 값

백엔드 환경 변수 목록

- coolsms: API KEY, API SECRET, 발신 번호

5. 배포하기

하나의 레포지토리에 여러 프로젝트가 묶여있을 경우 자동배포 구성

Name	Last commit	Last update
..		
📁 application-config	[CHORE] #169 DB 초기화 설정 변경	1 day ago
📁 config-service	[CHORE] #53 로컬 application.yml 배포 설정 삭제	2 weeks ago
📁 diary-service	[FEAT] #168 레포트 조회 시 자가진단 반환값 변경	1 day ago
📁 emergency-service	[FEAT] #151 긴급 메시지 보내기 로직 변경	3 days ago
📁 gateway-service	[FEAT] #151 토큰 검증 필터 제외 목록에 긴급메시지 보내는 url 추가	3 days ago
📁 panic-attack-service	[FEAT] 모델 반환값 로그 추가	2 days ago
📁 service-discovery	[CHORE] Docker File 수정	2 weeks ago
📁 user-service	[FEAT] #135 따소리 상호작용 동작 기능 구현	4 days ago
📁 voice-service	[FEAT] #197 목소리 샘플 스크립트 추가	1 hour ago

- 구축하고 싶었던 자동 배포 플로우
 1. develop 브랜치에서 master 브랜치로 병합
 2. master 브랜치를 기준으로 Docker 이미지를 생성
 3. 도커 이미지를 Docker Hub에 push
 4. EC2 서버에서 Docker Hub에 올라간 이미지를 pull
 5. EC2 서버에서 Docker 이미지 실행
- 하지만 내가 생각 했던 구성 방식에는 문제가 하나 있었다. 여러 프로젝트 중 하나의 프로젝트만 수정하더라도 모든 서버를 내렸다가 다시 올려야 한다는 것이었다.
- **"develop 브랜치에서 master 브랜치로 병합할 때 수정된 부분이 있는 프로젝트만 서버를 다시 띄우게 할 수는 없을까?"** 라는 아이디어에서 출발

파이프라인 설명

- git 명령어를 이용하여 **가장 최근에 발생한 merge 커밋의 커밋 아이디 hash 값**을 가져온다.

```
def mergeCommitHash = sh(  
    script: "git log -1 --merges --pretty=format:'%H'",  
    returnStdout: true  
) .trim()
```

- git diff 명령어를 이용하여 가장 **최근 병합 커밋에서 변경된 파일들의 목록**을 가져올 수 있다.

```
def changedFiles = sh(
    script: "git diff --name-only ${mergeCommitHash}^ ${mergeCommitHash}",
    returnStdout: true
).trim().split('\n')
```

```
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/web/CalendarController.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/web/DailyController.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/web/TrainingController.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/web/request/DailyRecordSaveRequest.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/web/request/TrainingRecordSaveRequest.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/web/response/CalendarResponse.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/web/response/CalendarsResponse.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/out/CalendarQueryAdapter.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/adapters/out/daily/DailyJpaEntity.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/application/service/DailyService.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/application/service/TrainingService.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/error/GlobalExceptionHandler.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/error/RecordDuplicateExceptionHandler.java
Backend/diary-service/src/main/java/com/ddasoom/diary_service/diary/error/TrainingTypeBadRequestException.java
```

Backend/emergency-service/Dockerfile
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/common/annotation/PersistenceAdapter.java
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/common/annotation/UseCase.java
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/common/annotation/WebAdapter.java
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/common/util/ApiUtils.java
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/emergency/adapters/in/web/PhoneBookController.java
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/emergency/adapters/in/web/request/AddPhoneRequest.java
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/emergency/adapters/in/web/response/PhoneBookResponse.java
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/emergency/adapters/out/PhoneBookAdapter.java
Backend/emergency-service/src/main/java/com/ddasoom/emergency_service/emergency/adapters/out/PhoneBookJpaEntity.java

...

Frontend/next.config.mjs
Frontend/package-lock.json
Frontend/package.json
Frontend/src/app/globals.css
Frontend/src/app/layout.tsx
Frontend/src/app/main/page.tsx

...

WebView/android/app/src/main/AndroidManifest.xml
WebView/android/app/src/main/java/com/ddasoom/wear/MainActivity.kt
WebView/android/app/src/main/java/com/ddasoom/wear/MainApp.kt

```

ication.kt
WebView/android/app/src/main/java/com/ddasoom/wear/ModuleTest.kt
WebView/android/app/src/main/java/com/ddasoom/wear/MyAppPackage.kt
WebView/android/app/src/main/java/com/ddasoom/wear/service/MessageService.kt

```

- 위 파일 목록에서 **변경된 프로젝트의 목록**을 알 수 있다. 각 파일 이름을 특정 길이까지 자르고 중복을 제거한다.
 - 백엔드는 Depth 2까지 (프로젝트가 여러개이기 때문)
 - 프론트와 안드로이드는 Depth 1까지

```

def changedProjects = changedFiles.collect { filePath ->
  def parts = filePath.split('/')

  if (filePath.startsWith('Backend')) {
    return "${parts[0]}/${parts[1]}"
  } else if (filePath.startsWith('Frontend')) {
    return "${parts[0]}"
  } else if (filePath.startsWith('WebView')) {
    return "${parts[0]}"
  } else {
    return filePath
  }
}.unique().collect { path ->
  return path.replace('/', '-')
}

```

```

Backend-diary-service
Backend-emergency-service
Frontend
WebView

```

- 위 코드에서 구한 `changedProjects` 리스트에 들어있는 프로젝트 이름과 파이프 라인의 이름을 동일하게 맞춰서 프로젝트당 하나씩 생성한다.

✓	☀	Backend-application-config
⌚	☀	Backend-config-service
✓	☀	Backend-diary-service
✓	☁☀	Backend-emergency-service
✓	☀	Backend-gateway-service
✓	☀	Backend-panic-attack-service
✓	☁	Backend-service-discovery
✓	☀	Backend-user-service
✓	☀	Backend-voice-service

- `changedProjects` 에 포함되어 있는 파이프라인을 비동기로 실행한다.
- 이 때 변경된 프로젝트에 `config-service` 가 포함되어 있는 경우 가장 먼저 실행시킨다. 서버를 빌드할 때 `config-service` 가 필요하기 때문이다.
- `cofing-service` 는 동기로 먼저 실행해서 다른 서버의 빌드 파이프라인이 실행될 때 먼저 `cofing-service` 가 떠있는 것이 보장되도록 한다.

```
if (parsedPaths.contains('Backend-config-service')) {
  build job: 'Backend-config-service', wait: true // 동기
  로 실행
  parsedPaths.remove('Backend-config-service')
}

changedProjects.each { pipelineName ->
  echo "Executing pipeline for: ${pipelineName}"
  try {
    build job: pipelineName, wait: false // 비동기로 실행
  } catch (Exception e) {
```



```

        echo "Pipeline '${pipelineName}' not found. Skipping...
    }
}

```

- 변경된 프로젝트만 골라서 새로 띄우는 파이프라인 작성 끝!

```

graph LR
    GITLAB["GITLAB"]
    GITLAB --> CD["CD"]
    CD --> BackendApplicationConfig["Backend-application-config"]
    CD --> BackendConfigService["Backend-config-service"]
    CD --> BackendDiaryService["Backend-diary-service"]
    CD --> BackendEmergencyService["Backend-emergency-service"]
    CD --> BackendGatewayService["Backend-gateway-service"]
    CD --> BackendPanicAttackService["Backend-panic-attack-service"]
    CD --> BackendServiceDiscovery["Backend-service-discovery"]
    CD --> BackendUserService["Backend-user-service"]
    CD --> BackendVoiceService["Backend-voice-service"]
    CD --> Frontend["Frontend"]

```

참고

- 각 서버별 파이프라인 순서는 다음과 같다. (예시: panic-attack-service)
 1. Git Clone
 2. Docker Hub 로그인
 3. Docker 이미지 빌드, push
 4. 서버 배포

```

pipeline {
    agent any

```

```

stages {
    stage('Git Clone') {
        steps {
            git branch: 'master', credentialsId: 'GITLAB_USER', url: 'https://lab.ssafy.com/s11-final/S11P31C103.git'
        }
        post {
            failure {
                echo 'Repository clone failure!'
            }
            success {
                echo 'Repository clone success!'
            }
        }
    }

    stage('Docker Hub Login') {
        steps {
            withCredentials([usernamePassword(credentialsId: 'DOCKER_USER', passwordVariable: 'DOCKER_PASSWORD', usernameVariable: 'DOCKER_USERNAME')]) {
                sh 'echo "$DOCKER_PASSWORD" | docker login -u $DOCKER_USERNAME --password-stdin'
            }
        }
    }

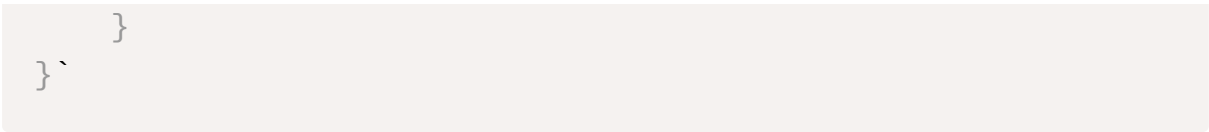
    stage('Docker Build and Push') {
        steps {
            sh 'cd ./Backend/panic-attack-service && docker build -f Dockerfile -t ddasoom/panic-attack-service .'
            sh 'cd ./Backend/panic-attack-service && docker push ddasoom/panic-attack-service'
            echo 'Docker image pushed successfully!'
        }
    }
}

```

```

stage('Deploy') {
    steps {
        script {
            // SSH 명령어를 사용할 때 문제가 발생할 수 있
            으니, `sshagent`를 사용하여 SSH 명령어 실행
            sshagent(credentials: ['SSH-CREDENTIAL
            S']) {
                // SSH 접속을 테스트하고 필요한 경우 SSH
                접속을 시도하는 명령어
                sh 'ssh -o StrictHostKeyChecking=no
                ubuntu@k11c103.p.ssafy.io "echo SSH connection successful"'
                sh 'ssh -o StrictHostKeyChecking=no
                ubuntu@k11c103.p.ssafy.io "sudo docker rm -f panic-attack-s
                ervice"'
                // 도커 컨테이너 실행
                sh """
                    ssh -o StrictHostKeyChecking=no
                    ubuntu@k11c103.p.ssafy.io \\\
                    'sudo docker pull ddasoom/panic
                    -attack-service && \\\
                    sudo docker run -d \\\
                    --name panic-attack-service
                    \\\
                    --network ddasoom-network
                    \\\
                    -e TZ=Asia/Seoul \\\
                    ddasoom/panic-attack-servic
                    e'
                    """
                // 사용하지 않는 Docker Image 정리
                sh 'docker image prune -f'
            }
        }
    }
}

```



▼ 배포 서버 환경 변수 설정

1. Jenkins 관리 클릭

S	W	Name	최근 성공	최근 실패	최근 소요 시간
✓	☀	Backend-application-config	4 days 6 hr #17	15 days #5	5.7 sec
⌚	☀	Backend-config-service	15 days #15	—	25 sec
✓	☀	Backend-diary-service	4 days 4 hr #30	14 days #15	1 min 8 sec
✓	☁	Backend-emergency-service	5 days 5 hr #31	8 days 3 hr #27	1 min 19 sec
✓	☀	Backend-gateway-service	5 days 5 hr #20	18 days #5	1 min 1 sec
✓	☀	Backend-panic-attack-service	5 days 4 hr #12	12 days #4	8 min 8 sec
✓	☁	Backend-service-discovery	15 days #8	18 days #5	31 sec
✓	☀	Backend-user-service	4 days 21 hr #25	17 days #15	30 sec
✓	☀	Backend-voice-service	2 days 6 hr #21	18 days #7	1 min 5 sec

2. Credentials 클릭

Security

- Security**
Secure Jenkins; define who is allowed to access/use the system.
- Credentials**
Configure credentials
- Credential Providers**
Configure the credential providers and types

Users

Create/delete/modify users that can log in to this Jenkins.

3. global 클릭

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	GITLAB_USER	enduf768640@gmail.com/*****
		System	(global)	DOCKER_USER	ddasoom103@gmail.com/*****
		System	(global)	SSH-CREDENTIALS	ubuntu
		System	(global)	MSG_KEY	coolsms api_key
		System	(global)	MSG_SECRET	coolsms api_secret
		System	(global)	MSG_SEND_NUMBER	sender phone number

4. +Add Credentials 클릭

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
GITLAB_USER	enduf768640@gmail.com/*****	Username with password	
DOCKER_USER	ddasoom103@gmail.com/*****	Username with password	
SSH-CREDENTIALS	ubuntu	SSH Username with private key	
MSG_KEY	coolsms api_key	Secret text	coolsms api_key
MSG_SECRET	coolsms api_secret	Secret text	coolsms api_secret
MSG_SEND_NUMBER	sender phone number	Secret text	sender phone number

5. 값 입력

New credentials

Kind
Username with password

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

Username ?

☐ Treat username as secret ?

Password ?

ID ?

Description ?

Create

6. 스크립트에서 credentials 값 불러와서 환경 변수 설정

Script ?

```
1 pipeline {
2     agent any
3
4     tools {
5         jdk 'jdk17'
6     }
7
8     environment {
9         // 환경 변수 선언
10        MSG_KEY = credentials('MSG_KEY')
11        MSG_SECRET = credentials('MSG_SECRET')
12        MSG_SEND_NUMBER = credentials('MSG_SEND_NUMBER')
13    }
14}
```

```
stage('Deploy') {
    steps {
        script {
            // SSH 명령어를 사용할 때 문제가 발생할 수 있으니, `sshagent`를 사용하여 SSH 명령어 실행
            sshagent(credentials: ['SSH-CREDENTIALS']) {
                // SSH 접속을 테스트하고 필요한 경우 SSH 접속을 시도하는 명령어
                sh 'ssh -o StrictHostKeyChecking=no ubuntu@k11c103.p.ssafy.io "echo SSH connection successful"'
                sh 'ssh -o StrictHostKeyChecking=no ubuntu@k11c103.p.ssafy.io "sudo docker rm -f emergency-service"'


                // 도커 컨테이너 실행
                sh """
                    ssh -o StrictHostKeyChecking=no ubuntu@k11c103.p.ssafy.io \\\
                    'sudo docker pull ddasoom/emergency-service && \\\
                    sudo docker run -d \\\
                    --name emergency-service \\\
                    --network ddasoom-network \\\
                    -e MSG_KEY=${MSG_KEY} \\\
                    -e MSG_SECRET=${MSG_SECRET} \\\
                    -e MSG_SEND_NUMBER=${MSG_SEND_NUMBER} \\\
                    -e TZ=Asia/Seoul \\\
                    ddasoom/emergency-service'
                """

                // 사용하지 않는 Docker Image 정리
                sh 'docker image prune -f'
            }
        }
    }
}
```

6. 빌드 및 실행

- Jenkinsfile 및 Dockerfile 확인
- WebView 프로젝트 apk를 빌드 및 실행
- wear 프로젝트를 adb/pairing으로 워치에 apk 설치해 실행
- 빌드 시 google-service.json 파일 확인 필요

서비스 별 DB 포트 정리

루트 계정은  계정 페이지에 정리되어 있습니다.

- user-service : 3306
- diary-service : 3307
- emergency-service : 3308
- voice-service : 3309

diary-service compose

```
services:
  mysql-diary:
    image: mysql
    container_name: mysql-diary
    restart: unless-stopped
    tty: true
    ports:
      - "3307:3306"
    environment:
      MYSQL_HOST: k11c103.p.ssafy.io
      MYSQL_ROOT_PASSWORD: ssafygwangjuc103
      MYSQL_DATABASE: ddasoom
      MYSQL_USER: ddasoom
      MYSQL_PASSWORD: ssafygwangjuc103
      SERVICE_TAGS: dev
      SERVICE_NAME: mysql
      TZ: Asia/Seoul
    volumes:
      - /home/ubuntu/mysql2_data:/var/lib/mysql # mysql
      - /home/ubuntu/mysql2_conf:/etc/mysql/conf.d # 추
```

주의사항!

- service 이름
- 컨테이너 이름
- 포트 번호

3 가지를 다르게 해야 함

kafka_compose.yml

```
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"
    networks:
      - ddasoom-network

  kafka:
    container_name: kafka
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "true"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    networks:
      - ddasoom-network

networks:
  ddasoom-network:
    external: true
```