포팅 매뉴얼

▼ 목차

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

🏠 사용한 JVM, 웹서버, WAS 제품 등의 종류와 설정 값, 버전(IDE 버전 포함) 기재

- **Back-End**

Java	17
Spring Boot	3.3.5
Redis	7.4.1
MySql	8.0
Nginx	1.27.2
Docker	27.3.1
Jenkins	2.478

- **Android**

Android	hedgehog
Kotlin	1.9.25

▼ 참고

```
agp = "8.5.2"
coilCompose = "2.4.0"
```

```
firebaseBom = "33.5.1"
glide_compose = "1.0.0-beta01"
kotlin = "1.9.25"
coreKtx = "1.13.1"
junit = "4.13.2"
junitVersion = "1.2.1"
espressoCore = "3.6.1"
appcompat = "1.7.0"
krossbowStompCore = "8.0.0"
lifecycleRuntimeCompose = "2.8.7"
material = "1.12.0"
hilt = "2.52"
hiltNavigationCompose = "1.2.0"
composeBom = "2024.10.00"
lifecycleRuntimeKtx = "2.8.6"
activityCompose = "1.9.3"
gson = "2.11.0"
pagingCompose = "3.3.2"
retrofit = "2.11.0"
okhttp = "4.12.0"
kotlinx-serialization = "1.7.3"
kotlinx-serialization-converter = "1.0.0"
kotlinxSerializationCore = "1.7.3"
coroutines = "1.9.0"
androidxDataStore = "1.1.1"
activity = "1.9.3"
constraintlayout = "2.1.4"
timber = "5.0.1"
accompanistSystemuicontroller = "0.34.0"
v2All = "2.20.0"
firebaseFirestoreKtx = "25.1.1"
lottie = "5.2.0"
firebaseMessagingKtx = "24.0.3"
moshi = "1.15.1"
```

🏠 빌드 시 사용되는 환경 변수 등의 내용 상세 기재

- **Back-End**
  - 각 서비스의 application.yml

```
spring:
  application:
    name: {서비스 이름}-service
  config:
    import: configserver:http://localhost:8888
  profiles:
    active: dev
```

  - discovery-service.yml

```
spring:
  application:
    name: discovery-service

eureka:
  client:
    fetch-registry: false
    register-with-eureka: false

server:
  port: 8761
```

  - api-gateway.yml

```
spring:
  application:
    name: api-gateway
  cloud:
    gateway:
      default-filters:
        - LoggingFilter
      routes:
        - id: user-service-public
          uri: lb://USER-SERVICE
          predicates:
```

```
            - Path=/users/login, /users/join, /users/jw

        - id: user-service-actuator
          uri: lb://USER-SERVICE
          predicates:
            - Path=/users/actuator/**
          filters:
            - StripPrefix=1

        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/users/**
          filters:
            - AuthorizationHeaderFilter

        - id: album-service-public
          uri: lb://ALBUM-SERVICE
          predicates:
            - Path=/albums/v3/api-docs

        - id: album-service-actuator
          uri: lb://ALBUM-SERVICE
          predicates:
            - Path=/albums/actuator/**
          filters:
            - StripPrefix=1

        - id: album-service
          uri: lb://ALBUM-SERVICE
          predicates:
            - Path=/albums/**
          filters:
            - AuthorizationHeaderFilter

        - id: calendar-service-public
          uri: lb://CALENDAR-SERVICE
          predicates:
```

```
        - Path=/calendars/v3/api-docs

    - id: calendar-service-actuator
      uri: lb://CALENDAR-SERVICE
      predicates:
        - Path= /calendars/actuator/**
      filters:
        - StripPrefix=1

    - id: calendar-service
      uri: lb://CALENDAR-SERVICE
      predicates:
        - Path=/calendars/**
      filters:
        - AuthorizationHeaderFilter

    - id: family-service-public
      uri: lb://FAMILY-SERVICE
      order: 1
      predicates:
        - Path=/family/v3/api-docs, /family/code/{c

    - id: web-socket-connect
      uri: lb:ws://FAMILY-SERVICE
      order: 2
      predicates:
        - Path=/family/ws-stomp/**
      filters:
        - StripPrefix=1

    - id: family-service-chat
      uri: lb://FAMILY-SERVICE
      order: 3
      predicates:
        - Path=/family/rooms/**, /family/actuator/*
      filters:
        - StripPrefix=1
```

```
        - id: family-service
          uri: lb://FAMILY-SERVICE
          order: 4
          predicates:
            - Path=/family/**
          filters:
            - AuthorizationHeaderFilter

        - id: interest-service-public
          uri: lb://INTEREST-SERVICE
          predicates:
            - Path=/interests/v3/api-docs

        - id: interest-service-actuator
          uri: lb://INTEREST-SERVICE
          predicates:
            - Path=/interests/actuator/**
          filters:
            - StripPrefix=1

        - id: interest-service
          uri: lb://INTEREST-SERVICE
          predicates:
            - Path=/interests/**
          filters:
            - AuthorizationHeaderFilter

        - id: notification-service-public
          uri: lb://NOTIFICATION-SERVICE
          predicates:
            - Path=/notifications/v3/api-docs

        - id: notification-service-actuator
          uri: lb://NOTIFICATION-SERVICE
          predicates:
            - Path=/notifications/actuator/**
          filters:
            - StripPrefix=1
```

```
      - id: notification-service
        uri: lb://NOTIFICATION-SERVICE
        predicates:
          - Path=/notifications/**
        filters:
          - AuthorizationHeaderFilter

      - id: question-service-public
        uri: lb://QUESTION-SERVICE
        predicates:
          - Path=/questions/v3/api-docs

      - id: question-service-actuator
        uri: lb://QUESTION-SERVICE
        predicates:
          - Path=/questions/actuator/**
        filters:
          - StripPrefix=1

      - id: question-service
        uri: lb://QUESTION-SERVICE
        predicates:
          - Path=/questions/**
        filters:
          - AuthorizationHeaderFilter

      - id: timecapsule-service-public
        uri: lb://TIMECAPSULE-SERVICE
        predicates:
          - Path=/timecapsules/v3/api-docs

      - id: timecapsule-service-actuator
        uri: lb://TIMECAPSULE-SERVICE
        predicates:
          - Path=/timecapsules/actuator/**
        filters:
          - StripPrefix=1
```

```
        - id: timecapsule-service
          uri: lb://TIMECAPSULE-SERVICE
          predicates:
            - Path=/timecapsules/**
          filters:
            - AuthorizationHeaderFilter

        - id: classification-service
          uri: lb://CLASSIFICATION-SERVICE
          predicates:
            - Path=/face-recognition/**

        - id: file-service-actuator
          uri: lb://FILE-SERVICE
          predicates:
            - Path=/files/actuator/**
          filters:
            - StripPrefix=1

        - id: file-service
          uri: lb://FILE-SERVICE
          predicates:
            - Path=/files/**

springdoc:
  swagger-ui:
    urls[0]:
      name: user-service
      url: /users/v3/api-docs
    urls[1]:
      name: calendar-service
      url: /calendars/v3/api-docs
    urls[2]:
      name: family-service
      url: /family/v3/api-docs
    urls[3]:
      name: interest-service
```

```
      url: /interests/v3/api-docs
    urls[4]:
      name: notification-service
      url: /notifications/v3/api-docs
    urls[5]:
      name: question-service
      url: /questions/v3/api-docs
    urls[6]:
      name: album-service
      url: /albums/v3/api-docs
    urls[7]:
      name: timecapsule-service
      url: /timecapsules/v3/api-docs
    urls[8]:
      name: classification-service
      url: /face-recognition/openapi.json

    use-root-path: true

server:
  port: 8000
```

- user-service.yml

```
spring:
  application:
    name: user-service

  datasource:
    url: jdbc:mysql://k11d108.p.ssafy.io:3307/d108?serv
    username: {DB-MYSQL-ID}
    password: {DB-MYSQL-PASSWORD}
    hikari:
      maximum-pool-size: 10

  data:
    redis:
      host: k11d108.p.ssafy.io
```

```
      port: 6379
      password: {DB-REDIS-PASSWORD}
  cache:
    type: redis

springdoc:
  swagger-ui:
    path: /users/swagger-ui.html
  api-docs:
    path: /users/v3/api-docs

  servlet:
    multipart:
      enabled: true
      max-file-size: 10MB
      max-request-size: 10MB

mybatis:
  mapper-locations: classpath:mappers/*.xml

feign:
  hystrix:
    enabled: true

server:
  port: 0
```

- family-service.yml

```
spring:
  application:
    name: family-service

  datasource:
    url: jdbc:mysql://k11d108.p.ssafy.io:3308/d108?serv
    username: {DB-MYSQL-ID}
    password: {DB-MYSQL-PASSWORD}
    hikari:
```

```
      maximum-pool-size: 10
      minimum-idle: 5
      idle-timeout: 600000
      max-lifetime: 1800000

  data:
    mongodb:
      uri: mongodb://{DB-MONGODB-ID}:{DB-MONGODB-PASSWO
    redis:
      host: k11d108.p.ssafy.io
      port: 6379
      password: {DB-REDIS-PASSWORD}

springdoc:
  swagger-ui:
    path: /family/swagger-ui.html
  api-docs:
    path: /family/v3/api-docs

websocket:
  server:
    url: ws://k11d108.p.ssafy.io:8765

mybatis:
  mapper-locations: classpath:mappers/*.xml
  configuration:
    map-underscore-to-camel-case: true
    default-timezone: Asia/Seoul

server:
  port: 0
```

- notification-service.yml

```
spring:
  application:
    name: notification-service
```

```
  # MySQL
  datasource:
    url: jdbc:mysql://k11d108.p.ssafy.io:3309/d108?serv
    username: {DB-MYSQL-ID}
    password: {DB-MYSQL-PASSWORD}
    hikari:
      maximum-pool-size: 10  # 커넥션 풀의 최대 크기 (필요에

springdoc:
  swagger-ui:
    path: /notifications/swagger-ui.html
  api-docs:
    path: /notifications/v3/api-docs

server:
  port: 0

firebase:
  path: familring-firebase-key.json

mybatis:
  mapper-locations: classpath:mappers/*.xml

feign:
  hystrix:
    enabled: true
```

- calendar-service.yml

```
spring:
  application:
    name: calendar-service

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://k11d108.p.ssafy.io:3311/d108?serv
    username: {DB-MYSQL-ID}
    password: {DB-MYSQL-PASSWORD}
```

```
    hikari:
      maximum-pool-size: 10

  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        jdbc:
          time_zone: Asia/Seoul
        default_batch_fetch_size: 100

aws:
  s3:
    daily-photo-path: daily

springdoc:
  swagger-ui:
    path: /calendars/swagger-ui.html
  api-docs:
    path: /calendars/v3/api-docs

server:
  port: 0
```

- album-service.yml

```
spring:
  application:
    name: album-service

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://k11d108.p.ssafy.io:3310/d108?serv
    username: {DB-MYSQL-ID}
    password: {DB-MYSQL-PASSWORD}
    hikari:
      maximum-pool-size: 10
```

```
  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        jdbc:
          time_zone: Asia/Seoul
        default_batch_fetch_size: 100

springdoc:
  swagger-ui:
    path: /albums/swagger-ui.html
  api-docs:
    path: /albums/v3/api-docs

aws:
  s3:
    album-photo-path: album

server:
  port: 0
```

- timecapsule-service.yml

```
spring:
  application:
    name: timecapsule-service

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://k11d108.p.ssafy.io:3313/d108?serv
    username: {DB-MYSQL-ID}
    password: {DB-MYSQL-PASSWORD}
    hikari:
      maximum-pool-size: 10

springdoc:
```

```
  swagger-ui:
    path: /timecapsules/swagger-ui.html
  api-docs:
    path: /timecapsules/v3/api-docs

server:
  port: 0
```

- question-service.ytml

```
spring:
  application:
    name: question-service

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://k11d108.p.ssafy.io:3312/d108?serv
    username: {DB-MYSQL-ID}
    password: {DB-MYSQL-PASSWORD}
    hikari:
      maximum-pool-size: 10

springdoc:
  swagger-ui:
    path: /questions/swagger-ui.html
  api-docs:
    path: /questions/v3/api-docs

server:
  port: 0
```

- interest-service.yml

```
spring:
  application:
    name: interest-service

  datasource:
```

```
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://k11d108.p.ssafy.io:3314/d108?serv
    username: {DB-MYSQL-ID}
    password: {DB-MYSQL-PASSWORD}
    hikari:
      maximum-pool-size: 10

aws:
  s3:
    interest-photo-path: interest

springdoc:
  swagger-ui:
    path: /interests/swagger-ui.html
  api-docs:
    path: /interests/v3/api-docs

server:
  port: 0
```

- file-service.yml

```
spring:
  application:
    name: file-service

server:
  port: 0
```

- **Android**
  - local.properties

```
SERVER_URL="https://k11d108.p.ssafy.io/"
KAKAO_API_KEY={KAKAO_API_KEY}
KAKAO_REDIRECT_URI="kakao47ed03b722abff7e166aef5b3c75c0
SOCKET_URL="https://k11d108.p.ssafy.io/family/ws-stomp/
```

🏠 배포 시 특이사항 기재

[MSA 활용]

: serivce, db 전부 micro 단위로 분리 (docker-compose 활용)

- **서비스 실행 명령어**

```
docker-compose -f docker-compose-service.yml up -d
```

- **데이터베이스 실행 명령어** (redis, mongo, mysql 활용)

```
docker-compose -f doker-compose-db.yml up -d
```

- **모니터링 및 Jenkins, Nginx 실행 명령어**

```
docker-compose -f docker-compose.yml up -d
```

- **서비스 별 DB 접속 포트**

```
user - 3307
family - 3308
notification - 3309
album - 3310
calendar - 3311
question - 3312
timecapsule - 3313
interest - 3314
```

[Jenkins Pipeline]

- **Master**

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
```

```
                checkout scm
            }
        }

        stage('Build and Deploy Config Server') {
            when {
                changeset "backend/config-service/**"
            }
            steps {
                withCredentials([file(credentialsId: 'conf
                    script {
                        dir('backend/config-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('config-s
                        }
                    }
                }
            }
        }

        stage('Build and Deploy Discovery Server') {
            when {
                changeset "backend/discovery-service/**"
            }
            steps {
                withCredentials([file(credentialsId: 'disc
                    script {
                        dir('backend/discovery-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('discover
                        }
                    }
                }
            }
        }

        stage('Build and Deploy API Gateway') {
            when {
```

```
                changeset "backend/api-gateway/**"
            }
            steps {
                withCredentials([file(credentialsId: 'api-
                    script {
                        dir('backend/api-gateway') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('api-gate
                        }
                    }
                }
            }
        }

        stage('Build and Deploy Services') {
            parallel {
                stage('Build and Deploy Classification Ser
                    steps {
                        script {
                            if (hasChanges('backend/classi
                                build job: 'classification
                            }
                        }
                    }
                }
                stage('Build and Deploy Album Service') {
                    steps {
                        script {
                            if (hasChanges('backend/album-
                                build job: 'album-service'
                            }
                        }
                    }
                }
                stage('Build and Deploy Calendar Service')
                    steps {
                        script {
                            if (hasChanges('backend/calend
```

```
                        build job: 'calendar-servi
                    }
                }
            }
        }
        stage('Build and Deploy User Service') {
            steps {
                script {
                    if (hasChanges('backend/user-s
                        build job: 'user-service',
                    }
                }
            }
        }
        stage('Build and Deploy Family Service') {
            steps {
                script {
                    if (hasChanges('backend/family
                        build job: 'family-service
                    }
                }
            }
        }
        stage('Build and Deploy File Service') {
            steps {
                script {
                    if (hasChanges('backend/file-s
                        build job: 'file-service',
                    }
                }
            }
        }
        stage('Build and Deploy Interest Service')
            steps {
                script {
                    if (hasChanges('backend/intere
                        build job: 'interest-servi
                    }
```

```
                        }
                    }
                }
                stage('Build and Deploy Notification Servi
                    steps {
                        script {
                            if (hasChanges('backend/notifi
                                build job: 'notification-s
                            }
                        }
                    }
                }
                stage('Build and Deploy Question Service')
                    steps {
                        script {
                            if (hasChanges('backend/questi
                                build job: 'question-servi
                            }
                        }
                    }
                }
                stage('Deploy Timecapsule Service') {
                    steps {
                        script {
                            if (hasChanges('backend/timeca
                                build job: 'timecapsule-se
                            }
                        }
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
```

```
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName, port) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d -p ${port}:${p

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}

// 변경 사항 확인 함수
def hasChanges(String path) {
    def changes = sh(script: "git diff --name-only HEAD^ H
    return changes == 0
}
```

- **Config**

```
pipeline {
    agent any

    stages {
```

```
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy Config Server') {
            steps {
                withCredentials([file(credentialsId: 'conf
                    script {
                        dir('backend/config-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('config-s
                        }
                    }
                }
            }
        }

        stage('Notification') {
            steps{
                echo 'jenkins notification!'
            }
            post {
                success {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME  //

                        // Mattermost로 성공 메시지 전송
                        mattermostSend(
                            color: 'good',
                            message: "✅ ${serviceName} 빌드
                        )
                    }
                }
```

```
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌드
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName, port) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
```

```
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d -p ${port}:${p

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Discovery**

```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy Discovery Server') {
            steps {
                withCredentials([file(credentialsId: 'disc
                    script {
                        dir('backend/discovery-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('discover
                        }
                    }
                }
            }
        }

        stage('Notification') {
            steps{
                echo 'jenkins notification!'
            }
            post {
```

```
                success {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME  //

                        // Mattermost로 성공 메시지 전송
                        mattermostSend(
                            color: 'good',
                            message: "✅ ${serviceName} 빌드
                        )
                    }
                }
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌드
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
```

```
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName, port) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d -p ${port}:${p

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Api-Gateway**

```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy API Gateway') {
            steps {
                withCredentials([file(credentialsId: 'api-
                    script {
                        dir('backend/api-gateway') {
```

```
                        copyApplicationYaml(applicatio
                        buildAndRunContainer('api-gate
                    }
                }
            }
        }
    }

    stage('Notification') {
        steps{
            echo 'jenkins notification!'
        }
        post {
            success {
                script {
                    // 마지막 커밋을 푸시한 사용자 정보 가져오
                    def user = sh(script: 'git log -1
                    def serviceName = env.JOB_NAME  //

                    // Mattermost로 성공 메시지 전송
                    mattermostSend(
                        color: 'good',
                        message: "✅ ${serviceName} 빌
                    )
                }
            }
            failure {
                script {
                    // 마지막 커밋을 푸시한 사용자 정보 가져오
                    def user = sh(script: 'git log -1
                    def serviceName = env.JOB_NAME

                    // Mattermost로 실패 메시지 전송
                    mattermostSend(
                        color: 'danger',
                        message: "❌ ${serviceName} 빌
                    )
                }
```

```
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName, port) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d -p ${port}:${p

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **User**

```
pipeline {
    agent any
```

```
    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy User Service') {
            steps {
                withCredentials([file(credentialsId: 'user
                    script {
                        dir('backend/user-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('user-ser
                        }
                    }
                }
            }
        }

        stage('Notification') {
            steps{
                echo 'jenkins notification!'
            }
            post {
                success {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME  //

                        // Mattermost로 성공 메시지 전송
                        mattermostSend(
                            color: 'good',
                            message: "✅ ${serviceName} 빌드
                        )
                    }
```

```
                }
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
```

```
    sh './gradlew build --build-cache -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}
}
```

- **Family**

```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy Family Service') {
            steps {
                withCredentials([file(credentialsId: 'fami
                    script {
                        dir('backend/family-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('family-s
                        }
                    }
                }
            }
        }

        stage('Notification') {
            steps{
                echo 'jenkins notification!'
            }
            post {
                success {
```

```
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME  //

                        // Mattermost로 성공 메시지 전송
                        mattermostSend(
                            color: 'good',
                            message: "✅ ${serviceName} 빌
                        )
                    }
                }
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
```

```
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Notification**

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy Notification Service') {
            steps {
                withCredentials([file(credentialsId: 'noti
                file(credentialsId: 'familring-firebase-ke
                ]) {
                    script {
```

```
                    dir('backend/notification-service'
                        copyApplicationYaml(applicatio

                        // firebase key 추가
                        sh "cp ${firebase_key} src/mai

                        buildAndRunContainer('notifica
                    }
                }
            }
        }
    }

    stage('Notification') {
        steps{
            echo 'jenkins notification!'
        }
        post {
            success {
                script {
                    // 마지막 커밋을 푸시한 사용자 정보 가져오
                    def user = sh(script: 'git log -1
                    def serviceName = env.JOB_NAME  //

                    // Mattermost로 성공 메시지 전송
                    mattermostSend(
                        color: 'good',
                        message: "✅ ${serviceName} 빌드
                    )
                }
            }
            failure {
                script {
                    // 마지막 커밋을 푸시한 사용자 정보 가져오
                    def user = sh(script: 'git log -1
                    def serviceName = env.JOB_NAME

                    // Mattermost로 실패 메시지 전송
```

```
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Calendar**

```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy Calendar Service') {
            steps {
                withCredentials([file(credentialsId: 'cale
                    script {
                        dir('backend/calendar-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('calendar
                        }
                    }
                }
            }
        }

        stage('Notification') {
            steps{
                echo 'jenkins notification!'
            }
            post {
                success {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME  //

                        // Mattermost로 성공 메시지 전송
```

```
                        mattermostSend(
                            color: 'good',
                            message: "✅ ${serviceName} 빌
                        )
                    }
                }
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}
```

```
// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Album**

```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy Album Service') {
            steps {
                withCredentials([file(credentialsId: 'albu
                    script {
                        dir('backend/album-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('album-se
                        }
                    }
                }
            }
```

```
        }

        stage('Notification') {
            steps{
                echo 'jenkins notification!'
            }
            post {
                success {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME  //

                        // Mattermost로 성공 메시지 전송
                        mattermostSend(
                            color: 'good',
                            message: "✅ ${serviceName} 빌드
                        )
                    }
                }
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌드
                        )
                    }
                }
            }
        }
    }
}
```

```
// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Classification**

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
```

```
        }

        stage('Build and Deploy Classification Service') {
            steps {
                withCredentials([file(credentialsId: 'clas
                    script {
                        dir('backend/classification-servic
                            copyApplicationEnv(ENV_FILE)
                            buildAndRunContainer('classifi
                        }
                    }
                }
            }
        }
    }
}

// 공통 함수
def copyApplicationEnv(env) {
    sh "cp ${env} .env"
    sh "cat .env"
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${container}"
        sh "docker rm ${container}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)

    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} --env-file .env -
```

```
    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Timecapsule**

```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy TimeCapsule Service') {
            steps {
                withCredentials([file(credentialsId: 'time
                    script {
                        dir('backend/timecapsule-service')
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('timecaps
                        }
                    }
                }
            }
        }

        stage('Notification') {
            steps{
                echo 'jenkins notification!'
            }
            post {
                success {
```

```
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME  //

                        // Mattermost로 성공 메시지 전송
                        mattermostSend(
                            color: 'good',
                            message: "✅ ${serviceName} 빌
                        )
                    }
                }
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
```

```
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Quesiton**

```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy Question Service') {
            steps {
                withCredentials([file(credentialsId: 'ques
                    script {
                        dir('backend/question-service') {
```

```
                        copyApplicationYaml(applicatio
                        buildAndRunContainer('question
                    }
                }
            }
        }
    }

    stage('Notification') {
        steps{
            echo 'jenkins notification!'
        }
        post {
            success {
                script {
                    // 마지막 커밋을 푸시한 사용자 정보 가져오
                    def user = sh(script: 'git log -1
                    def serviceName = env.JOB_NAME  //

                    // Mattermost로 성공 메시지 전송
                    mattermostSend(
                        color: 'good',
                        message: "✅ ${serviceName} 빌
                    )
                }
            }
            failure {
                script {
                    // 마지막 커밋을 푸시한 사용자 정보 가져오
                    def user = sh(script: 'git log -1
                    def serviceName = env.JOB_NAME

                    // Mattermost로 실패 메시지 전송
                    mattermostSend(
                        color: 'danger',
                        message: "❌ ${serviceName} 빌
                    )
                }
```

```
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **Interest**

```
pipeline {
    agent any
```

```
stages {

    stage('Checkout') {
        steps {
            checkout scm
        }
    }

    stage('Build and Deploy Interest Service') {
        steps {
            withCredentials([file(credentialsId: 'inte
                script {
                    dir('backend/interest-service') {
                        copyApplicationYaml(applicatio
                        buildAndRunContainer('interest
                    }
                }
            }
        }
    }

    stage('Notification') {
        steps{
            echo 'jenkins notification!'
        }
        post {
            success {
                script {
                    // 마지막 커밋을 푸시한 사용자 정보 가져오
                    def user = sh(script: 'git log -1
                    def serviceName = env.JOB_NAME  //

                    // Mattermost로 성공 메시지 전송
                    mattermostSend(
                        color: 'good',
                        message: "✅ ${serviceName} 빌드
                    )
```

```
                    }
                }
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌드
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName) {
    stopAndRemoveContainer(serviceName)
```

```
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

- **File**

```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build and Deploy File Service') {
            steps {
                withCredentials([file(credentialsId: 'file
                    script {
                        dir('backend/file-service') {
                            copyApplicationYaml(applicatio
                            buildAndRunContainer('file-ser
                        }
                    }
                }
            }
        }

        stage('Notification') {
            steps{
```

```
                echo 'jenkins notification!'
            }
            post {
                success {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME  //

                        // Mattermost로 성공 메시지 전송
                        mattermostSend(
                            color: 'good',
                            message: "✅ ${serviceName} 빌
                        )
                    }
                }
                failure {
                    script {
                        // 마지막 커밋을 푸시한 사용자 정보 가져오
                        def user = sh(script: 'git log -1
                        def serviceName = env.JOB_NAME

                        // Mattermost로 실패 메시지 전송
                        mattermostSend(
                            color: 'danger',
                            message: "❌ ${serviceName} 빌
                        )
                    }
                }
            }
        }
    }
}

// 공통 함수: application.yml 복사
def copyApplicationYaml(application_yml) {
    sh 'mkdir -p src/main/resources'
    sh "cp ${application_yml} src/main/resources/applicati
```

```
}

// 공통 함수: 컨테이너 중지 및 삭제
def stopAndRemoveContainer(containerName) {
    def container = sh(script: "docker ps -a --filter 'nam
    if (container) {
        sh "docker stop ${containerName}"
        sh "docker rm ${containerName}"
    }
}

// 공통 함수: 컨테이너 빌드 및 실행
def buildAndRunContainer(serviceName, port = null) {
    stopAndRemoveContainer(serviceName)
    sh 'chmod +x ./gradlew'
    sh './gradlew clean build -x test'
    sh "docker build -t ${serviceName} ."
    sh "docker run --name ${serviceName} -d ${serviceName}

    // 사용하지 않는 Docker 이미지 정리
    sh "docker image prune -f"
}
```

2. 프로젝트에 사용되는 외부 서비스 정리를 정리한 문서

- **Kako Login API**

3. DB 덤프 파일 최신본

- **User**

```
create database d108;
use d108;

-- DROP
DROP TABLE IF EXISTS `user`;

-- USER
CREATE TABLE `user` (
    `user_id`   BIGINT  NOT NULL AUTO_INCREMENT,
    `user_kakao_id` VARCHAR(100)    NOT NULL COMMENT '탈퇴
    `user_password` VARCHAR(255)    NOT NULL COMMENT '탈퇴
    `user_nickname` VARCHAR(100)    NOT NULL COMMENT '탈퇴
    `user_birth_date`   DATE    NOT NULL    COMMENT '띠 계산
    `user_zodiac_sign`  VARCHAR(255)    NOT NULL    COMMEN
    `user_role` ENUM('F', 'M', 'S', 'D', 'N')   NOT NULL
    `user_face` VARCHAR(255)    NOT NULL    COMMENT '얼사분
    `user_color`    VARCHAR(100)    NOT NULL    COMMENT 'H
    `user_emotion`  VARCHAR(100)    NOT NULL    DEFAULT "탈
  `user_fcm_token` VARCHAR(255) NULL COMMENT '탈퇴 시 빈 문자
  `user_un_read_count` INT NOT NULL DEFAULT 0 COMMENT '회원
    `user_created_at`   TIMESTAMP   NOT NULL    DEFAULT NO
    `user_modified_at`  TIMESTAMP   NOT NULL    DEFAULT NO
  `user_is_lunar` TINYINT NOT NULL DEFAULT 0,
    `user_is_deleted`   TINYINT NOT NULL    DEFAULT 0,
    `user_is_admin` TINYINT NOT NULL    DEFAULT 0,
    PRIMARY KEY(`user_id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='회원 정보를 
```

- **Family**

```
create database d108;
use d108;

-- DROP
DROP TABLE IF EXISTS `family_user`;
DROP TABLE IF EXISTS `family`;
```

```
-- FAMILY
CREATE TABLE `family` (
    `family_id` BIGINT  NOT NULL AUTO_INCREMENT,
    `family_code`   CHAR(6) NOT NULL,
    `family_count`  TINYINT NOT NULL,
    `family_created_at` TIMESTAMP   NOT NULL,
    `family_communication_status`   INT NOT NULL    DEFAUL
    PRIMARY KEY(`family_id`)
);

CREATE TABLE `family_user` (
    `family_user_id`    BIGINT  NOT NULL AUTO_INCREMENT,
    `family_id` BIGINT  NOT NULL,
    `user_id`   BIGINT  NOT NULL,
    PRIMARY KEY(`family_user_id`),
    CONSTRAINT `fk_family_id_family-user` FOREIGN KEY(`fam
);
```

- **Notification**

```
create database d108;
use d108;

-- DROP
DROP TABLE IF EXISTS `notification`;

-- NOTIFICATION
CREATE TABLE `notification` (
    `notification_id` BIGINT NOT NULL AUTO_INCREMENT,
    `receiver_user_id` BIGINT NOT NULL COMMENT '알림을 받을 
    `sender_user_id` BIGINT NULL COMMENT '알림을 발생시킨 사용
    `destination_id` VARCHAR(100) NULL COMMENT '이동할 페이지
    `notification_type` ENUM('KNOCK', 'MENTION_CHAT', 'MEN
    `notification_title` VARCHAR(50) NOT NULL,
    `notification_message` VARCHAR(100) NOT NULL,
    `notification_is_read` TINYINT NOT NULL DEFAULT FALSE,
    `notification_read_at` TIMESTAMP NULL,
```

```
    `notification_created_at` TIMESTAMP NOT NULL DEFAULT N
    PRIMARY KEY(`notification_id`)
);
```

- **Album**

```
create database d108;
use d108;

-- DROP
DROP TABLE IF EXISTS `photo`;
DROP TABLE IF EXISTS `album`;

-- ALBUM
CREATE TABLE `album` (
    `album_id`  BIGINT  NOT NULL PRIMARY KEY AUTO_INCREMEN
    `family_id` BIGINT  NOT NULL,
    `user_id`   BIGINT  NULL,
    `schedule_id`   BIGINT  NULL,
    `album_name`    VARCHAR(100)    NOT NULL,
    `album_type`    VARCHAR(20) NOT NULL
);


CREATE TABLE `photo` (
    `photo_id`  BIGINT  NOT NULL PRIMARY KEY AUTO_INCREMEN
    `parent_photo_id` BIGINT NULL,
    `album_id`  BIGINT  NOT NULL,
    `photo_url` VARCHAR(255)    NOT NULL,
    FOREIGN KEY (`album_id`) REFERENCES `album` (`album_id
    FOREIGN KEY (`parent_photo_id`) REFERENCES `photo` (`p
);
```

- **Calendar**

```
create database d108;
use d108;
```

```
-- DROP
DROP TABLE IF EXISTS `schedule_user`;
DROP TABLE IF EXISTS `schedule`;
DROP TABLE IF EXISTS `daily`;

-- CALENDAR
CREATE TABLE `schedule` (
    `schedule_id`   BIGINT  NOT NULL PRIMARY KEY AUTO_INCR
    `family_id` BIGINT  NOT NULL,
    `schedule_start_time`   TIMESTAMP   NOT NULL,
    `schedule_end_time` TIMESTAMP   NOT NULL,
    `schedule_title`    VARCHAR(100)    NOT NULL,
    `schedule_has_notification` TINYINT NOT NULL,
    `schedule_has_time` TINYINT NOT NULL,
    `schedule_color`    VARCHAR(100)    NOT NULL
);


CREATE TABLE `schedule_user` (
    `schedule_user_id`  BIGINT  NOT NULL PRIMARY KEY AUTO_
    `schedule_id`   BIGINT  NOT NULL,
    `attendee_id`   BIGINT  NOT NULL,
    `schedule_user_attendance_status`   TINYINT NOT NULL,
    FOREIGN KEY (`schedule_id`) REFERENCES `schedule` (`sc
);


CREATE TABLE `daily` (
    `daily_id`  BIGINT  NOT NULL PRIMARY KEY AUTO_INCREMEN
    `family_id` BIGINT  NOT NULL,
    `author_id` BIGINT  NOT NULL,
    `daily_content` VARCHAR(255)    NOT NULL,
    `daily_photo_url`   VARCHAR(255)    NOT NULL,
    `daily_created_at`  TIMESTAMP   NOT NULL    DEFAULT NO
    `daily_modified_at` TIMESTAMP   NOT NULL    DEFAULT NO
);
```

- **Question**

```
create database d108;
use d108;

-- DROP
DROP TABLE IF EXISTS `question_answer`;
DROP TABLE IF EXISTS `question_family`;
DROP TABLE IF EXISTS `question`;

-- QUESTION
CREATE TABLE `question` (
    `question_id`   BIGINT  NOT NULL AUTO_INCREMENT,
    `question_content`  VARCHAR(255)    NOT NULL,
    PRIMARY KEY(`question_id`)
);

CREATE TABLE `question_family` (
    `question_family_id`    BIGINT  NOT NULL AUTO_INCREMEN
    `question_id`   BIGINT  NOT NULL,
    `family_id` BIGINT  NOT NULL,
    PRIMARY KEY(`question_family_id`),
    CONSTRAINT `fk_question_id_question-family` FOREIGN KE
);

CREATE TABLE `question_answer` (
    `question_answer_id`    BIGINT  NOT NULL AUTO_INCREMEN
    `question_family_id`    BIGINT  NOT NULL,
    `user_id`   BIGINT  NOT NULL    COMMENT '질문 답변 작성자
    `question_answer`   VARCHAR(255)    NOT NULL,
    `question_created_at`   TIMESTAMP   NOT NULL,
    `question_modified_at`  TIMESTAMP   NOT NULL,
    PRIMARY KEY(`question_answer_id`),
    CONSTRAINT `fk_question_family_id_question-answer` FOR
);

INSERT INTO `question` (`question_content`) VALUES
("오늘 가장 인상 깊었던 일은 무엇인가요?"),
("최근 본 영화나 드라마 중 가장 추천하고 싶은 것은 무엇인가요?"),
("가족 중 누군가의 별명을 만든다면 뭐라고 하고 싶나요?"),
```

```
("가장 좋아하는 계절은 언제인가요? 이유는 무엇인가요?"),
("최근에 가장 크게 웃었던 일은 무엇인가요?"),
("어릴 적 자주 했던 놀이가 있다면?"),
("최근 읽은 책 중 인상 깊었던 구절은 무엇인가요?"),
("지금 집에 가장 필요한 물건이 있다면 무엇일까요?"),
("어제 저녁 메뉴 중 가장 맛있었던 것은?"),
("가족 여행을 떠난다면 어디로 가고 싶나요?"),
("가장 기억에 남는 가족 여행지는?"),
("내가 가장 좋아하는 음식은 무엇인가요?"),
("가족 중 요리를 가장 잘한다고 생각하는 사람은?"),
("만약 동물을 키운다면 어떤 동물을 키우고 싶나요?"),
("어릴 적 장래 희망은 무엇이었나요?"),
("지금 가장 가고 싶은 나라는 어디인가요?"),
("이번 주말에 하고 싶은 계획이 있다면?"),
("내가 생각하는 가족의 가장 큰 장점은?"),
("내가 생각하는 가족의 가장 큰 약점은?"),
("오늘 나에게 가장 감사했던 사람은 누구인가요?"),
("가장 좋아하는 노래나 음악 장르는?"),
("좋아하는 배우나 가수가 있다면?"),
("최근 새롭게 알게 된 사실이 있나요?"),
("나에게 가장 행복한 순간은 언제인가요?"),
("어릴 적 가장 좋아했던 만화 캐릭터는?"),
("가족 중 요리 대결을 한다면 이길 사람은 누구일까요?"),
("내가 가장 좋아하는 색깔은 무엇인가요?"),
("가족 중 패션 감각이 가장 뛰어나다고 생각하는 사람은?"),
("내가 최근 배운 것 중 가장 유용한 것은 무엇인가요?"),
("지금 나에게 가장 필요한 것은 무엇일까요?"),
("좋아하는 운동이나 취미가 있나요?"),
("어릴 적 꿈꿨던 상상 속 직업은 무엇인가요?"),
("평생 하나의 음식을 먹어야 한다면 어떤 음식을 선택할 건가요?"),
("가족 중 가장 기억에 남는 생일은 누구의 생일인가요?"),
("내가 생각하는 가족 간의 이상적인 하루는 어떤 모습인가요?"),
("혼자 있는 시간이 필요할 때 주로 무엇을 하나요?"),
("내가 가장 존경하는 사람은 누구인가요?"),
("이웃 중에 친하게 지내는 사람이 있나요?"),
("언젠가 꼭 해보고 싶은 버킷리스트가 있다면?"),
("가족 중 가장 유머러스한 사람은 누구라고 생각하나요?"),
("어릴 적 가장 재미있었던 기억이 있다면?"),
```

```
("가족 중 가장 정리정돈을 잘하는 사람은 누구인가요?"),
("가장 좋아하는 과일은 무엇인가요?"),
("집에서 가장 좋아하는 공간은 어디인가요?"),
("나만의 스트레스 해소 방법이 있다면?"),
("내가 가장 좋아하는 시간대는 언제인가요?"),
("최근 이루어낸 작은 성취가 있다면 무엇인가요?"),
("지금 집에 있는 식물이나 반려동물이 있다면?"),
("내가 가족에게 바라는 점이 있다면?"),
("요즘 가장 관심 있는 뉴스나 이슈가 있다면?"),
("좋아하는 동물이 있다면 무엇인가요?"),
("가족과 함께 하고 싶은 스포츠가 있다면?"),
("집에서 가장 자주 사용하는 물건은 무엇인가요?"),
("최근 가장 많이 하는 생각은 무엇인가요?"),
("좋아하는 배우가 있다면 누구인가요?"),
("가족 중 가장 기억에 남는 추억은 무엇인가요?"),
("친구들과 자주 가는 장소가 있다면?"),
("지금 나의 일상에서 감사한 일은 무엇인가요?"),
("최근 본 가장 아름다운 풍경은 무엇인가요?"),
("가족 중 가장 먼저 기상하는 사람은?"),
("나의 하루 중 가장 기대되는 순간은 언제인가요?"),
("가족 간의 약속이 있다면 지켜야 할 중요한 것은 무엇일까요?"),
("가족 중 주말에 가장 활발하게 활동하는 사람은?"),
("가장 좋아하는 계절에 하고 싶은 활동이 있다면?"),
("가족 중 휴일에 주로 무엇을 하는 사람이 있나요?"),
("가족이 함께 하는 기념일이 있다면 언제인가요?"),
("내가 가족을 위해 해주고 싶은 일이 있다면?"),
("가족 중 누가 가장 재치가 넘친다고 생각하나요?"),
("주말에 가족과 함께 즐길 수 있는 취미가 있다면?"),
("아침을 먹는 스타일은 어떤가요?"),
("가장 좋아하는 꽃이 있다면 무엇인가요?"),
("내가 가족을 위해 바라는 것은 무엇인가요?"),
("내가 생각하는 가족의 의미는 무엇인가요?"),
("오늘 하루 가장 고마웠던 일은 무엇인가요?"),
("가족 중 누군가에게 특별히 전하고 싶은 말이 있다면?"),
("최근 가족을 위해 만든 요리가 있다면?"),
("내가 가족에게 받은 가장 큰 사랑은 무엇인가요?"),
("가족과 함께 나누고 싶은 취미가 있다면?"),
("나에게 가족은 어떤 존재인가요?"),
```

```
("가족 중 가장 꼼꼼하게 계획을 세우는 사람은 누구인가요?"),
("집에서 가족이 함께하는 시간을 어떻게 보내고 싶은가요?"),
("가족이 함께 떠나는 휴가가 있다면 어디로 가고 싶나요?"),
("아침에 가장 먼저 하는 일은 무엇인가요?"),
("가족에게 전하고 싶은 오늘의 감사 인사는?"),
("내가 가족을 위해 해주고 싶은 것은 무엇인가요?"),
("지금 나에게 가장 행복한 일은 무엇인가요?"),
("어렸을 때 꿈꾸던 미래는 어떤 모습이었나요?"),
("가족 중 가장 친한 사람은 누구인가요?"),
("최근에 새로 배우고 싶은 것이 있다면 무엇인가요?"),
("오늘 하루 중 가장 의미 있었던 일은 무엇인가요?"),
("가족에게 듣고 싶은 칭찬이 있다면?"),
("요즘 나의 가장 큰 관심사는 무엇인가요?"),
("가족과 함께 했던 웃긴 기억이 있다면?"),
("나의 취미를 가족과 공유해 본 적이 있나요?"),
("내가 가장 소중하게 생각하는 물건은 무엇인가요?"),
("가장 좋아하는 영화나 드라마의 장르는?"),
("내가 가족과 더 소통하고 싶은 이유는?"),
("가족이 다 함께 먹고 싶은 음식이 있다면?"),
("가족 중 가장 여유로운 성격을 가진 사람은 누구인가요?"),
("최근 가족과 함께 나누고 싶은 좋은 소식이 있다면?"),
("내가 가족과 하고 싶은 스포츠 활동은?"),
("가족과 공유하고 싶은 일상 사진이 있다면?"),
("가족 중 가장 독특한 취미를 가진 사람은?"),
("나의 성격을 잘 나타내는 색깔이 있다면?"),
("가족이 함께 도전할 만한 새로운 활동이 있다면?"),
("오늘 하루를 다시 떠올리며 한 마디로 표현한다면?"),
("가장 좋아하는 음료는 무엇인가요?"),
("가족과의 소소한 일상이 나에게 어떤 의미를 주나요?"),
("내가 가장 좋아하는 휴식 시간은 언제인가요?"),
("오늘 하루 중 가장 행복했던 순간은 언제인가요?"),
("가족에게 자랑하고 싶은 나의 장점은?"),
("가족과의 대화에서 가장 자주 나오는 주제는?"),
("가족 중 여행을 좋아하는 사람이 있다면 어디로 자주 가나요?"),
("가족 간에 함께 만들어가고 싶은 전통이 있다면?"),
("가족과 함께 하고 싶은 명절 음식은 무엇인가요?"),
("나의 하루 중 가장 편안한 시간은 언제인가요?"),
("가족 중 가장 호기심이 많은 사람은 누구인가요?"),
```

```
("가족과 함께 가보고 싶은 박물관이나 전시회가 있다면?"),
("가족과 함께 보낸 시간이 나에게 주는 의미는?"),
("가족 중 가장 고생이 많은 사람에게 하고 싶은 말은?"),
("가족에게 줄 선물을 고민한다면 무엇을 하고 싶나요?"),
("가장 좋아하는 가족 행사나 기념일이 있다면?"),
("오늘 가장 많이 웃었던 순간은 언제인가요?"),
("가족과 함께 하고 싶은 여가 활동이 있다면?"),
("나의 하루 중 가장 감사한 순간은 언제인가요?"),
("가족과 함께 나누고 싶은 오늘의 생각은?"),
("가족에게 전하고 싶은 나의 작은 성공 이야기는?"),
("가족과 함께 보고 싶은 영화나 드라마가 있다면?"),
("가족 중 가장 활기차고 긍정적인 사람은 누구인가요?"),
("나의 하루 중 가장 충전이 필요한 시간은 언제인가요?"),
("요즘 새롭게 도전하고 싶은 것이 있다면 무엇인가요?"),
("오늘 하루 중 가장 뿌듯했던 일은 무엇인가요?"),
("가족과 함께 나누고 싶은 추억이 있다면 무엇인가요?"),
("최근 내가 해보고 싶은 일은 무엇인가요?"),
("가족과 함께 가고 싶은 유명한 맛집이 있나요?"),
("내가 자주 쓰는 유행어나 말투가 있다면 무엇인가요?"),
("어렸을 때 좋아했던 놀이가 있다면?"),
("내가 최근에 배운 신기한 사실은?"),
("가족 중 취미가 가장 다양한 사람은 누구인가요?"),
("가장 좋아하는 간식이나 디저트는?"),
("내가 생각하는 가족 중 최고의 요리사는 누구인가요?"),
("나만의 특별한 휴식 방법이 있다면?"),
("요즘 가장 즐거웠던 시간은 언제인가요?"),
("나의 하루 중 가장 피곤한 시간대는 언제인가요?"),
("가족과 함께 도전하고 싶은 여행지가 있다면?"),
("가족과 함께 보낼 수 있는 하루가 주어진다면 무엇을 하고 싶나요?"),
("내가 자주 찾는 동네 명소는 어디인가요?"),
("가족과 함께 한 가장 기억에 남는 이벤트는 무엇인가요?"),
("내가 가족과 나누고 싶은 이야기가 있다면?"),
("최근에 새롭게 생긴 나만의 습관이 있나요?"),
("가족 중 가장 사랑받는 취미는 무엇인가요?"),
("나의 하루 중 가장 생각이 많은 시간대는 언제인가요?"),
("최근 가족과 공유하고 싶은 취미가 있다면?"),
("내가 가장 자주 듣는 음악은 어떤 장르인가요?"),
("가족과 함께 즐길 수 있는 게임이 있다면?"),
```

```
("나에게 가족이란 어떤 의미인가요?"),
("내가 가족에게 자주 묻는 질문은 무엇인가요?"),
("요즘 가장 행복한 순간은 언제인가요?"),
("가족과 함께 기억에 남는 사진을 찍은 장소는?"),
("내가 꿈꾸는 집이 있다면 어떤 모습인가요?"),
("요즘 나의 하루 중 가장 즐거운 순간은 언제인가요?"),
("가족과 함께 꾸준히 해보고 싶은 활동이 있나요?"),
("가장 기억에 남는 특별한 날은 언제였나요?"),
("내가 가장 좋아하는 주말 활동은 무엇인가요?"),
("요즘 나의 생활에서 가장 필요한 변화가 있다면 무엇인가요?"),
("가족과 함께 해보고 싶은 취미가 있다면?"),
("내가 자주 가는 산책로는 어디인가요?"),
("가족에게 가장 고마운 순간은 언제였나요?"),
("내가 좋아하는 계절에 가족과 하고 싶은 일이 있다면?"),
("최근에 나에게 좋은 영향을 준 사람은 누구인가요?"),
("나에게 가족이란 어떤 안정감을 주나요?"),
("요즘 가장 관심 있는 건강 관리 방법이 있다면?"),
("내가 가족과 함께 먹고 싶은 음식은?"),
("가족과 함께 해보고 싶은 모험이 있다면?"),
("내가 가장 좋아하는 운동 종목은 무엇인가요?"),
("나의 일상 속 가장 소중한 시간은 언제인가요?"),
("가족 중 가장 자주 웃는 사람은 누구인가요?"),
("나의 일상 속 소중한 습관은 무엇인가요?"),
("가족과 함께 가고 싶은 전시회가 있다면?"),
("가족과 함께 맞이하고 싶은 명절은 무엇인가요?"),
("나의 하루 중 가장 여유로운 시간대는 언제인가요?"),
("가족과 함께 공유하고 싶은 명언이 있다면?"),
("내가 가장 좋아하는 간식 시간은 언제인가요?"),
("가족과 함께 나누고 싶은 최근의 기쁨은?"),
("가족 중 가장 세심한 성격을 가진 사람은 누구인가요?"),
("요즘 나의 생활에서 가장 즐거운 점은 무엇인가요?"),
("가족과 함께 할 수 있는 운동이 있다면?"),
("나에게 가족과 함께 보내는 시간이 주는 의미는?"),
("요즘 내가 가장 좋아하는 공간은 어디인가요?"),
("가족과 함께 듣고 싶은 음악이 있다면?"),
("내가 자주 즐기는 간식은 무엇인가요?"),
("가족과 함께 가고 싶은 유명한 산이나 바다도시가 있다면?"),
("내가 가족에게 전하고 싶은 따뜻한 말이 있다면?"),
```

```
("가족과 함께 나누고 싶은 여행 이야기가 있다면?"),
("내가 가족과 더 많은 시간을 보내고 싶은 이유는?"),
("가족과 함께 시작하고 싶은 새로운 취미는?"),
("최근에 가장 좋았던 책이나 영화는 무엇인가요?"),
("나의 하루 중 가장 큰 즐거움은 무엇인가요?"),
("가족과 함께 나누고 싶은 오늘의 목표는 무엇인가요?"),
("오늘 나에게 가장 감사한 순간은 언제인가요?");
```

- **Timecapsule**

```
create database d108;
use d108;

-- DROP
DROP TABLE IF EXISTS `timecapsule_answer`;
DROP TABLE IF EXISTS `timecapsule`;

-- TIMECAPSULE
CREATE TABLE `timecapsule` (
    `timecapsule_id`    BIGINT  NOT NULL AUTO_INCREMENT,
    `family_id` BIGINT  NOT NULL,
    `timecapsule_start_date`    TIMESTAMP   NOT NULL,
    `timecapsule_end_date`  TIMESTAMP   NOT NULL,
    PRIMARY KEY(`timecapsule_id`)
);

CREATE TABLE `timecapsule_answer` (
    `timecapsule_answer_id` BIGINT  NOT NULL AUTO_INCREMEN
    `timecapsule_id`    BIGINT  NOT NULL,
    `timecapsule_answer_content`    VARCHAR(255)    NOT NU
    `timecapsule_answer_created_at` TIMESTAMP   NOT NULL,
    `user_id` BIGINT NOT NULL COMMENT '타임캡슐 답변 작성자 고
    PRIMARY KEY(`timecapsule_answer_id`),
    CONSTRAINT `fk_timecapsule_id_timecapsule-anwser` FORE
);
```

- **Interest**

```
create database d108;
use d108;

-- DROP
DROP TABLE IF EXISTS `notification`;

-- NOTIFICATION
CREATE TABLE `notification` (
    `notification_id` BIGINT NOT NULL AUTO_INCREMENT,
    `receiver_user_id` BIGINT NOT NULL COMMENT '알림을 받을 사
    `sender_user_id` BIGINT NULL COMMENT '알림을 발생시킨 사용
    `destination_id` VARCHAR(100) NULL COMMENT '이동할 페이지
    `notification_type` ENUM('KNOCK', 'MENTION_CHAT', 'MENT
    `notification_title` VARCHAR(50) NOT NULL,
    `notification_message` VARCHAR(100) NOT NULL,
    `notification_is_read` TINYINT NOT NULL DEFAULT FALSE,
    `notification_read_at` TIMESTAMP NULL,
    `notification_created_at` TIMESTAMP NOT NULL DEFAULT NO
    PRIMARY KEY(`notification_id`)
);
```