Next-Generation Content Platform

Dailyfeed
Backend Architecture

주요 백엔드 기능 및 설계 관련 설명

2026/01/09 (정순구)

핵심 설계 원칙 (Core Principles)

Kafka 에러/장애 처리 지원

kafka 작업 실패 시 실패 작업에 대한 deadletter 처리 메커니즘 지원

스케일아웃 용도별 서비스 그룹 분리

쓰기 vs 통계/조회 용도별 분리. 트래픽 급증 시 특정 기능만 스케일 아웃 가능한 구조 지향.

결합도 분리

통신 레벨(L4)과 애플리케이션 레벨(L7)의 분리를 통해 독립적인 모듈성 유지.

Kafka Publisher,Listener 통신실패 시 메시지 배치처리 Rule

(1) dead letter 기록 후 배치 처리

publisher
- kafka_publisher_dead_letters

listener
- kafka_listener_dead_letters

(2) dead letter 기록 실패 시에는 통신 레벨의 실패로 간주

- offset 커밋 x
- 트랜잭션 롤백

Read/Write 용도로 애플리케이션 스케일 아웃 그룹을 분리

Read 부하가 증가할 경우
- timeline 그룹의 스케일 아웃을 진행

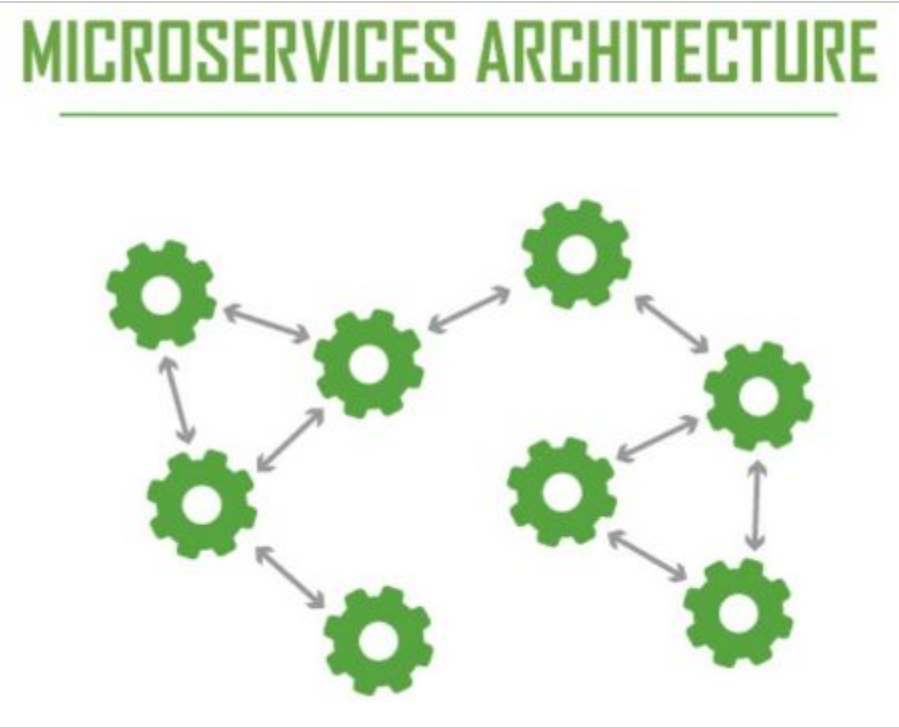
Write 부하가 증가할 경우
- content 그룹의 스케일 아웃을 진행

Feign, KafkaListener 내의 코드의 통신/애플리케이션 로직 분리

- Kafka Listener 내에서 메시지의 중복여부 판단 로직, 저장 로직 등이 결합된 스파게티 코드 자제
- Feign 레벨(HTTP)의 오류와 애플리케이션의 오류를 분리
- e.g. -Helper 클래스 : 통신 레벨 코드 담당

Overview

- **member-svc:** 회원가입, 로그인, 인증 관리
- **content-svc:** 글/댓글 생성 및 수정 (Write 전용)
- **timeline-svc:** 팔로우 피드, 인기글 조회 (Read 전용)
- **search-svc:** Full-Text Search 전용 서비스
- **activity-svc:** 사용자 활동 기록 저장 및 처리
- **image-svc:** 이미지 업로드 및 프로필 관리
- **batch-svc:** dead letter 처리



Dailyfeed Backend Service Overview

member-svc

- 회원가입, 로그인, 로그아웃
- 사용자 인증 및 권한 관리
- 프로필 요약 정보 제공

Replicas: 1 ~ 2

MySQL

content-svc

- 글 작성 / 수정 / 삭제
- 댓글 및 답글 CRUD
- 좋아요 / 좋아요 취소 처리

Replicas: 1 ~ 3

MySQL Mongo

image-svc

- 프로필 / 썸네일 업로드
- 이미지 조회 및 삭제
- 나의 프로필 이미지 관리

Replicas: 3 ~ 5

PVC

activity-svc

- 사용자 활동 기록 저장
- 활동 로그 조회 및 수정
- Kafka 이벤트 메시징 처리

Logging Service

Mongo

timeline-svc

- 팔로우 피드 (Follow Feed)
- 인기글 / 댓글 많은 글 통계
- 사용자 맞춤 피드 조회

Replicas: 1 ~ 7

MySQL Mongo

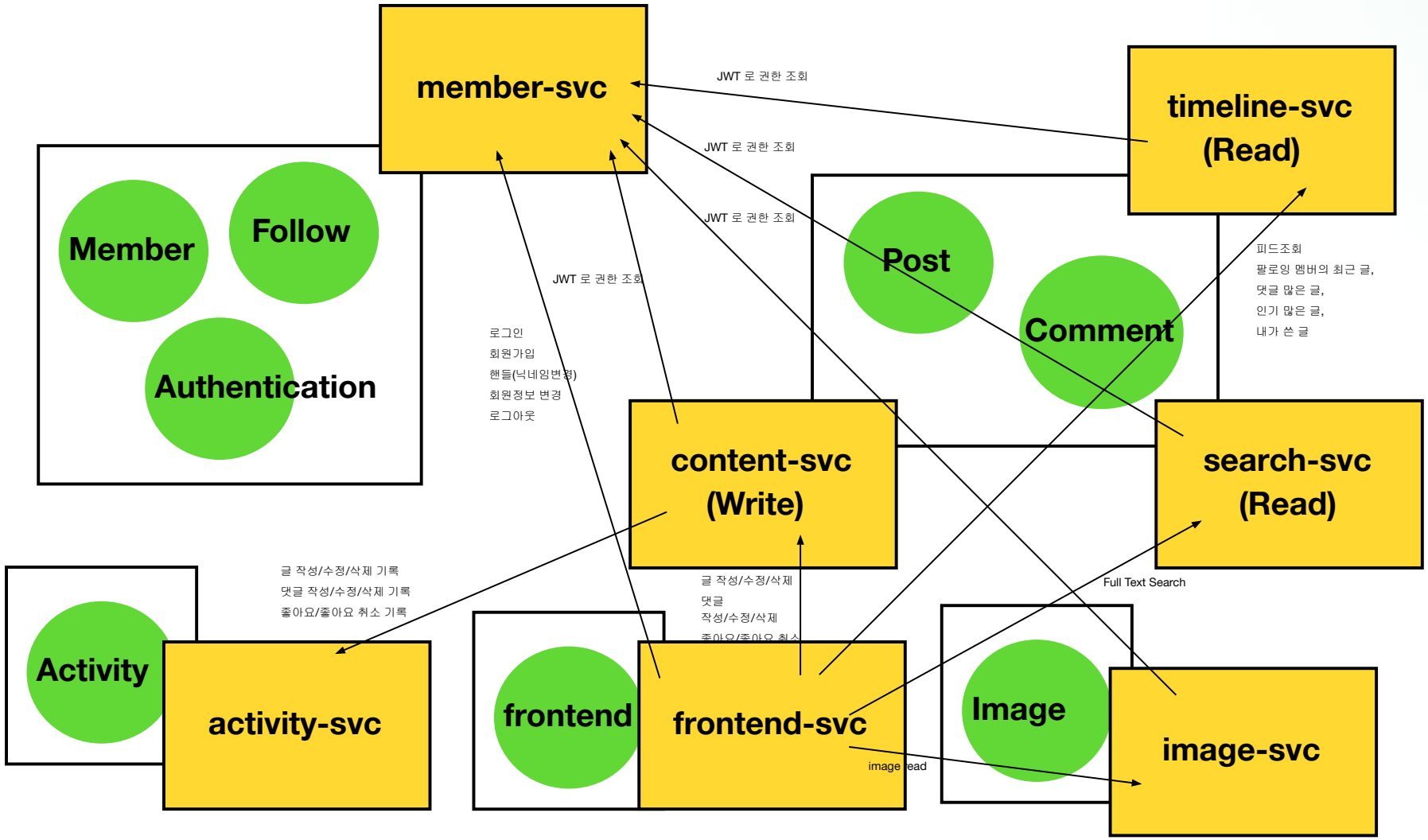
search-svc

- 본문 Full-Text Search
- 형태소 분석 기반 검색
- 검색 특화 별도 인스턴스

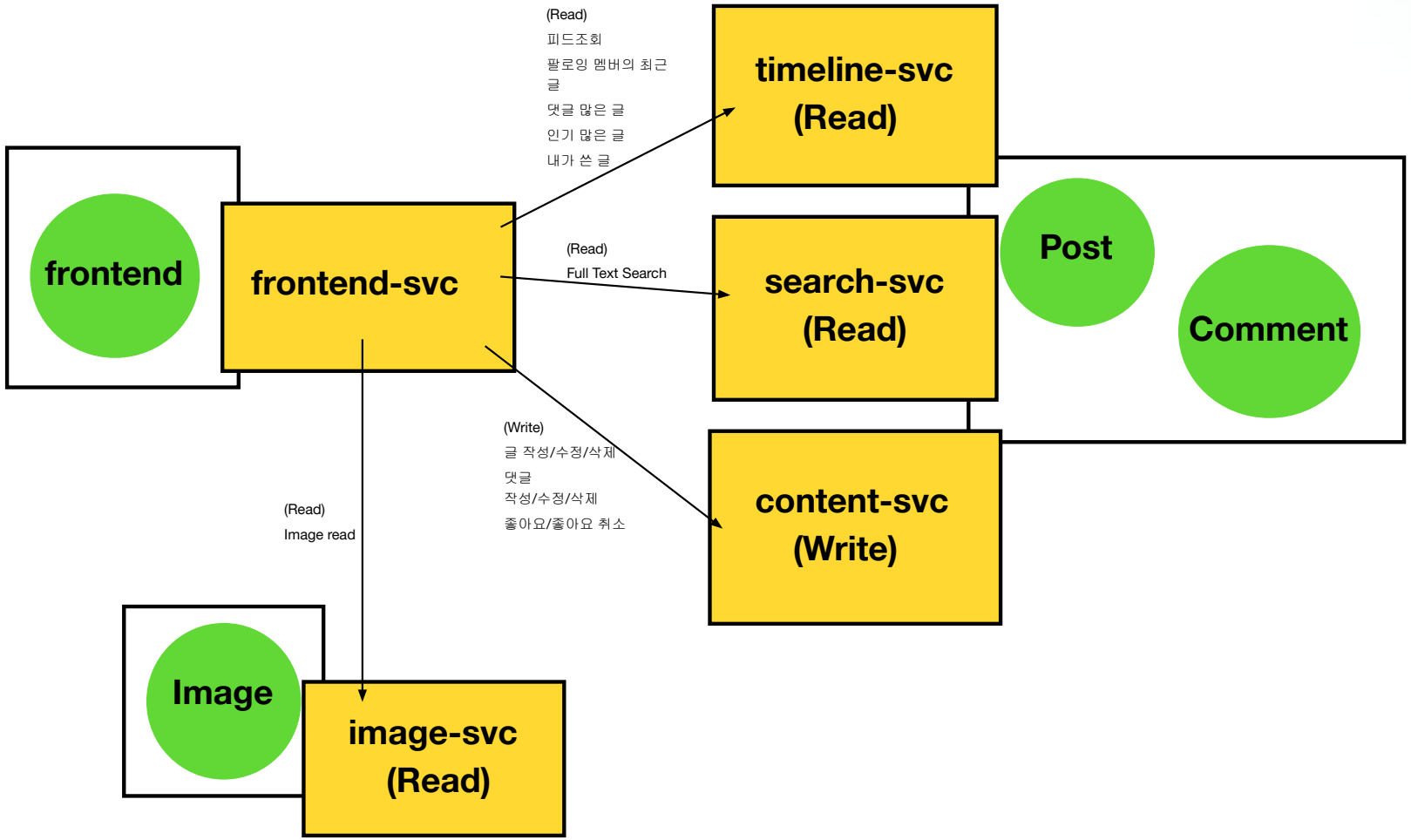
Replicas: 1 ~ 3

Mongo

도메인 경계 (전체)


member-svc
timeline-svc
(Read)
JWT 로 권한 조회
JWT 로 권한 조회
JWT 로 권한 조회
JWT 로 권한 조회
Member
Follow
Authentication
Post
Comment
피드조회
팔로잉 멤버의 최근 글,
댓글 많은 글,
인기 많은 글,
내가 쓴 글
로그인
회원가입
핸들(닉네임변경)
회원정보 변경
로그아웃
content-svc
(Write)
search-svc
(Read)
Full Text Search
글 작성/수정/삭제 기록
댓글 작성/수정/삭제 기록
좋아요/좋아요 취소 기록
글 작성/수정/삭제
댓글
작성/수정/삭제
Activity
activity-svc
frontend
frontend-svc
Image
image-svc
image read

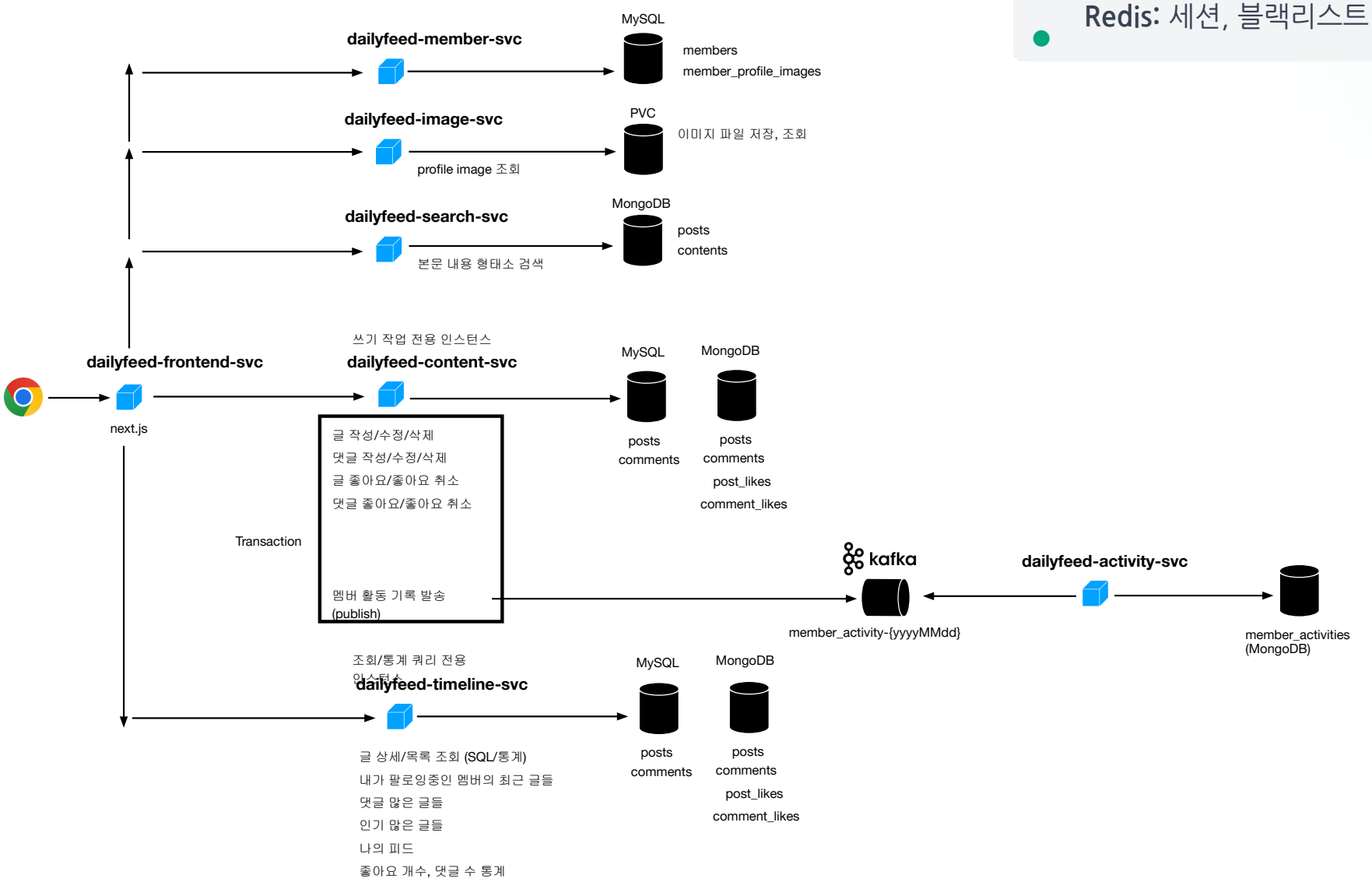
도메인 경계 (Post, Comment, Image)



전반적인 흐름

Storage

- **MySQL:** 관계형 데이터 (Member, Post)
- **MongoDB:** Full Text Search 용도의 데이터, dead letter
- **Redis:** 세션, 블랙리스트



스케일아웃 그룹 분류 방식 소개

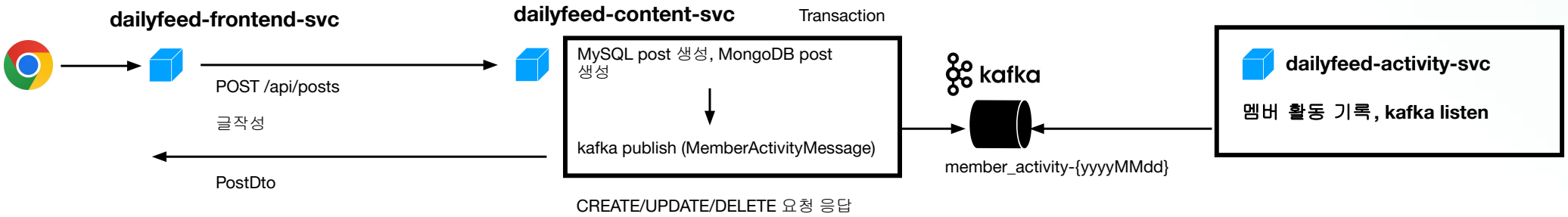
스케일아웃 그룹 분류

- Read 스케일아웃 그룹
- Create/Update/Delete 스케일아웃 그룹

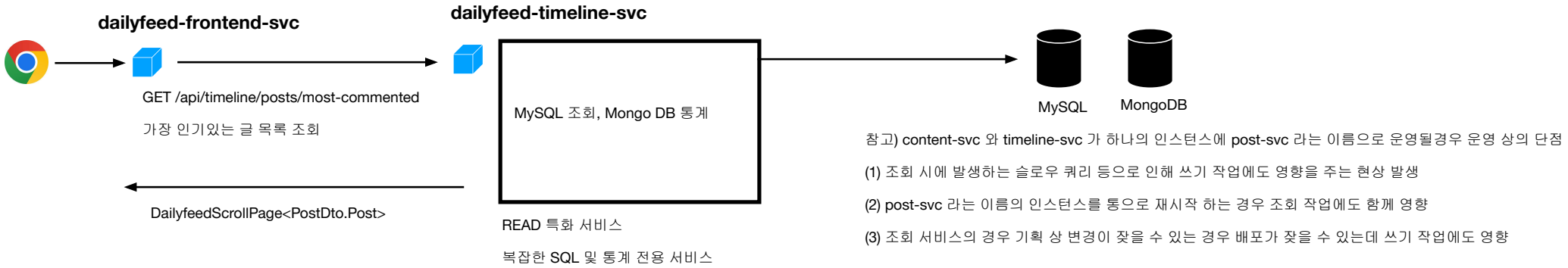
kafka

스케일 아웃 그룹 분류

Read 스케일 아웃 그룹



Create/Update/Delete 스케일 아웃 그룹



참고) content-svc 와 timeline-svc 가 하나의 인스턴스에 post-svc 라는 이름으로 운영될경우 운영 상의 단점

(1) 조회 시에 발생하는 슬로우 쿼리 등으로 인해 쓰기 작업에도 영향을 주는 현상 발생

(2) post-svc 라는 이름의 인스턴스를 통으로 재시작 하는 경우 조회 작업에도 함께 영향

(3) 조회 서비스의 경우 기획 상 변경이 잦을 수 있는 경우 배포가 잦을 수 있는데 쓰기 작업에도 영향

JWT 인증 방식

dailyfeed-member-svc 인증방식

- Key Refresh
- Blacklist, Logout
- AccessToken, RefreshToken
  - accessToken, refreshToken
  - login
  - accessToken 만료 시 인증 흐름
  - refreshToken 만료 시 인증 흐름
- ArgumentResolver

dailyfeed-member-svc 소개

- **Key Refresh**
  - JWT 를 생성하는 dailyfeed-member-svc 의 서버 Key 는 유효기간을 가지고 있으며 계속해서 변경됩니다.
  - 클라이언트의 요청 시 서버 Key 만료된 시점일 경우 서버는 응답헤더로 {X-Token-Refresh-Needed: 'true'} 와 함께 401 UnAuthorized 를 응답합니다.
  - 클라이언트는 하던 작업을 잠시 중단 후 서버에 POST /api/token/refresh 를 통해 새로운 accessToken, refreshToken 을 획득합니다.
  - 클라이언트는 새로운 accessToken, refreshToken 으로 하던 중단 했던 작업을 재개합니다.

- **Blacklist, Logout**
  - 로그아웃을 했을 때 로그아웃된 해당 토큰을 Blacklist 에 추가해서, 중복된 토큰을 사용하지 않도록 보장합니다. 만료된 토큰,로그아웃된 토큰을 악용하는 케이스에 대응할수 있습니다.
  - 만약 Blacklist 에 해당되는 토큰으로 API 요청이 발생할 경우 서버에서는 401 UnAuthorized 를 응답하며 응답헤더로 {'X-Relogin-Required': 'true'} 를 응답해서 새로 다시 로그인하도록 유도, 클라이언트는 401 UnAuthorized, 응답헤더 {'X-Relogin-Required': 'true'} 를 응답받으면 sessionStorage, localStorage 를 무효화하고, /login 페이지로 사용자를 리다이렉트 시킵니다.

- **accessToken, refreshToken**
  - accessToken : 인증이 정상적으로 이뤄졌을 때 클라이언트가 유효한 사용자임을 보장하는 JWT
  - refreshToken : accessToken 만료시 refreshToken 을 통해 accessToken 을 새로 refresh 하도록 유도, refreshToken 이 만료되었을 경우에는 재로그인 하도록 안내

dailyfeed-member-svc 소개

- **ArgumentResolver**
  - ArgumentResolver 의 역할로 충분하기에 ArgumentResolver 를 선택
  - MemberProfileArgumentResolver : member-svc 가 아닌 다른 백엔드에서 feign 을 통해 member-svc 로 인증여부,유효성 체크시 사용하는 역할을 수행
  - AuthenticatedMember**Internal**ArgumentResolver : member-svc 에서 요청의 인증 유효성을 체크하는 데에 사용하는 ArgumentResolver

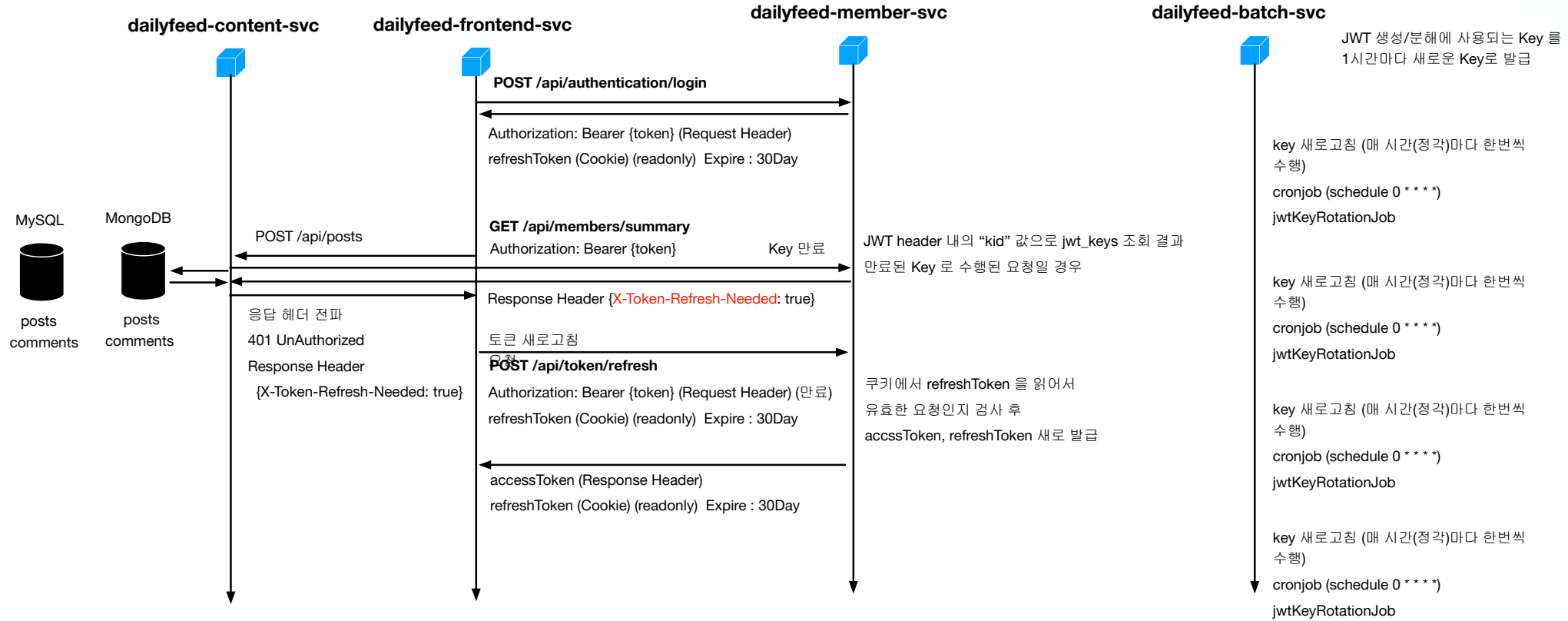
- **AOP, Interceptor 대신 ArgumentResolver 를 채택한 이유**
  - AOP 를 통한 인증 방식 대신 ArgumentResolver 를 채택, 웹 요청/엔드포인트에 대한 파라미터 검증의 경우 ArgumentResolver 가 적합하기 때문
  - 지나친 AOP 사용은 불필요한 곳 까지 해당 AOP 기능이 고르게 잠식되어 독이 되는 케이스도 다수 존재하는 점을 고려
  - ArgumentResolver 의 역할로 충분하기에 ArgumentResolver 를 선택
  - Interceptor 를 사용하지 않은 이유는 인증기능이 API 엔드포인트의 경로와 네이밍에 의존적으로 되기에 배제

JWT Key Refresh

Key Refresh

- JWT Header 내의 “kid” 값으로 “jwt_keys” 테이블 조회 후, 찾은 Key 가 만료된 Key(is_active=false)일 경우
- 응답 헤더 {X-Token-Refresh-Needed: true} 와 함께 401 UnAuthorized 응답
- 클라이언트는 해당 작업을 잠시 멈춰둔 채로 서버로 POST /api/token/refresh 요청을 보내 새로운 accessToken, refreshToken 을 부여받습니다.
- 클라이언트는 새로운 accessToken, refreshToken 으로 하던 작업을 재개

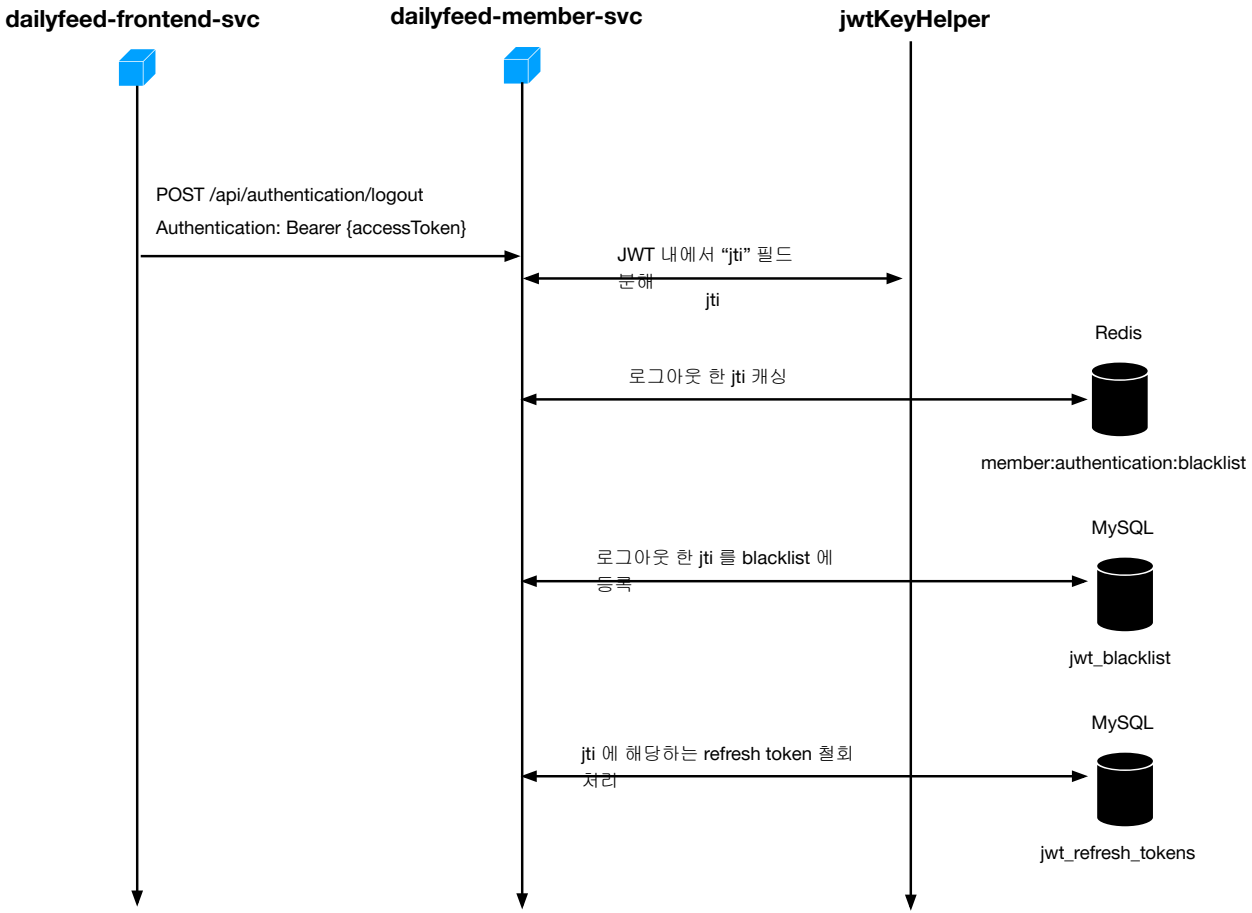
(참고) JWT 를 생성하는 Key 는 유효기간을 가지고 있으며 계속해서 변경되며, 1시간에 1번씩 cronjob으로 수행되는 dailyfeed-batch-svc 에 의해 새로운 key 로 변경되며 ‘jwt_keys’ 테이블에 저장됩니다.



Blacklist, Logout

Blacklist 처리 프로세스

- 로그아웃 시 로그아웃한 토큰은 Blacklist 에 추가해서 중복된 토큰을 사용하지 않도록 보장해서 만료된 토큰,로가아웃된 토큰을 악용하는 케이스에 대응
- Blacklist 에 해당하는 토큰으로 API 요청 발생시 서버에서는 응답헤더 {'X-Relogin-Required': 'true} 와 함께 401 UnAuthorized 를 응답
- 클라이언트는 sessionStorage, localStorage 를 무효화하고, /login 페이지로 사용자를 리다이렉트 시킴



accessToken, refreshToken

accessToken

- 사용자가 인증되었음을 의미하는 JWT
- 짧은 만료기한을 가지며, AccessToken 이 만료되었을 경우 응답헤더 {'X-Token-Refresh-Needed': 'true'} 와 함께 401 UnAuthorized 을 응답

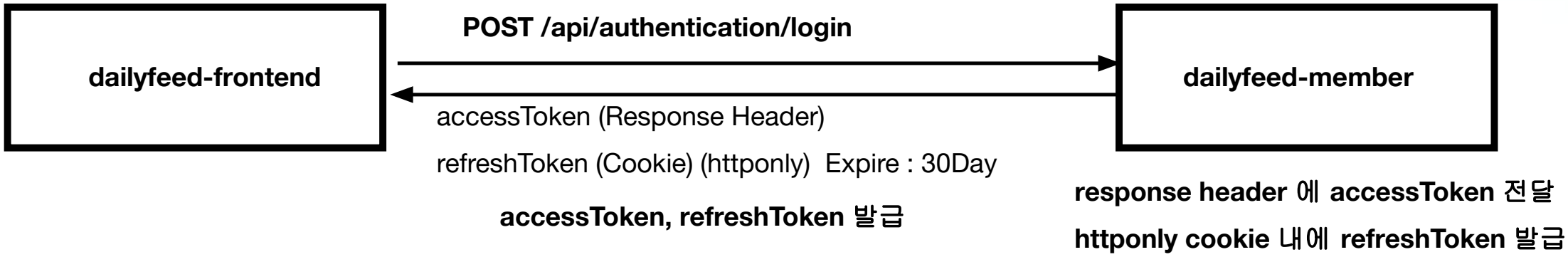
refreshToken

- 새로운 accessToken, refreshToken 을 부여받을 때 사용하는 토큰
- refreshToken 이 만료되었을 경우 서버에서 {X-Token-Relogin-Required: true} 응답헤더와 함께 401 UnAuthorized 를 응답
- 즉, refreshToken 이 만료된 경우는 새로 로그인을 시도하도록 유도

AccessToken, RefreshToken - login

login 프로세스

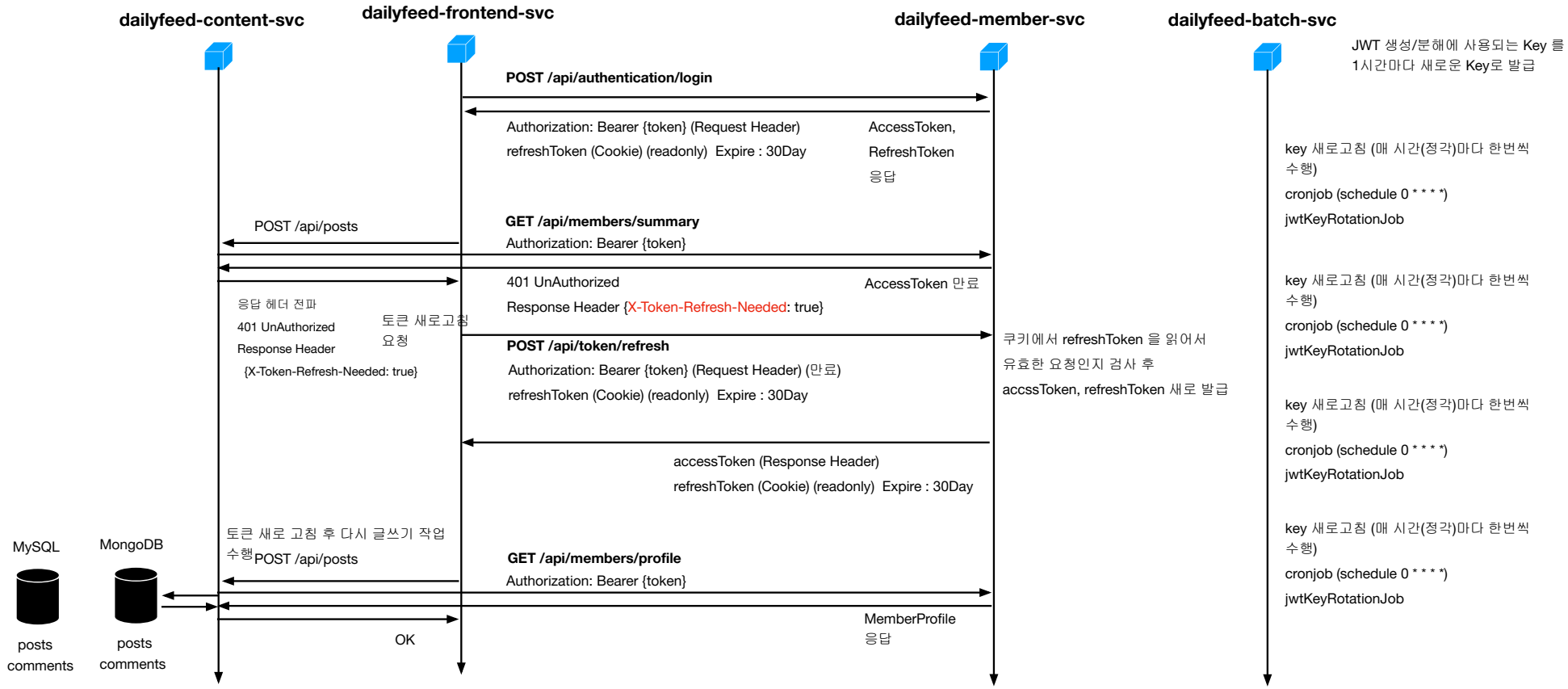
- 클라이언트에서 POST /api/authentication/login 요청
- 서버는 응답헤더에 accessToken 을 담고, HTTPOnly Cookie 에 refreshToken 을 담아서 200 을 응답 (accessToken, refreshToken 발급)



AccessToken, RefreshToken - accessToken 만료시

accessToken 만료시 accessToken을 새로 발급받는 절차

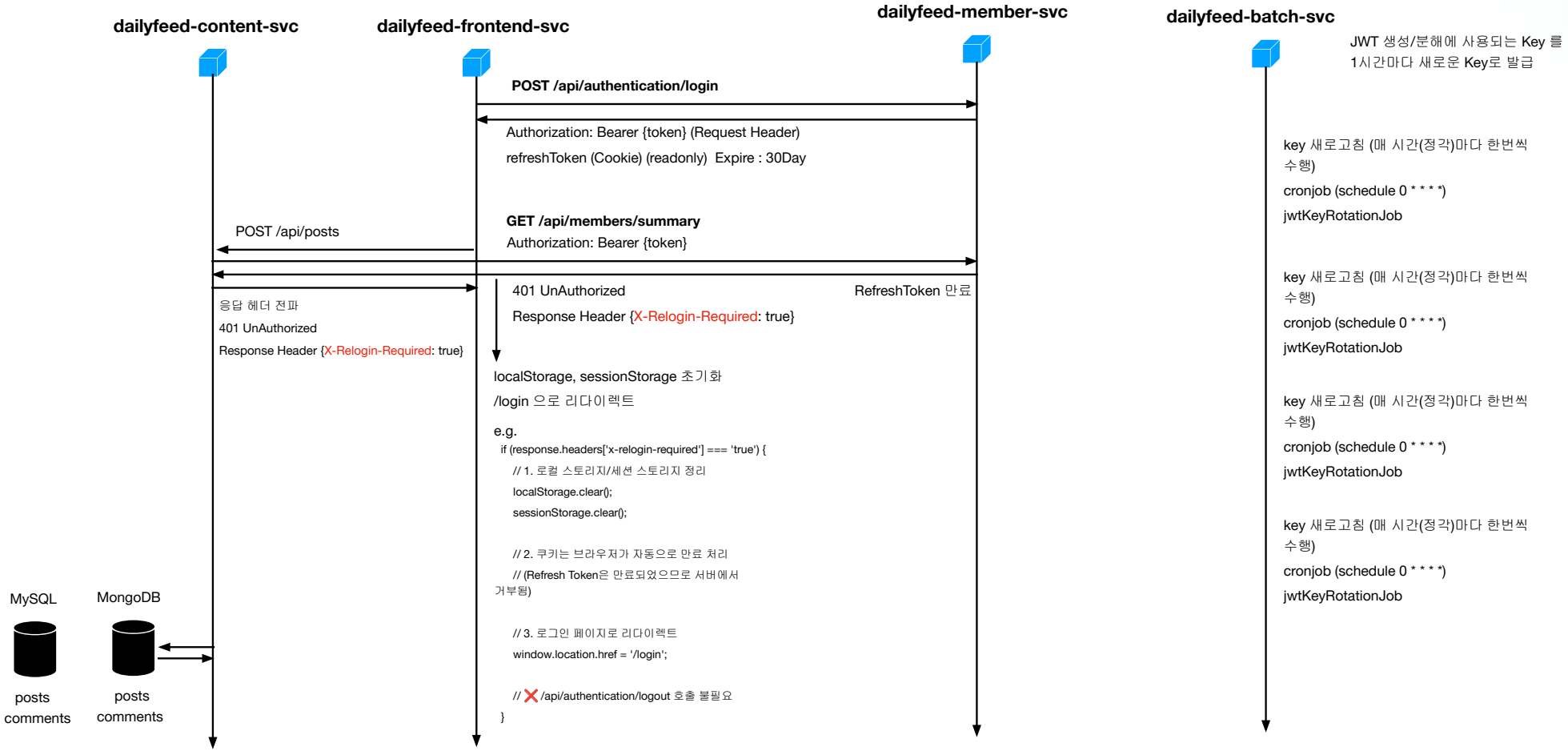
- 클라이언트의 특정 작업 수행 도중 accessToken 만료 (서버에서 {X-Token-Refresh-Needed: true} 응답헤더와 함께 401 UnAuthorized 를 응답)
- 클라이언트는 하던 작업을 멈추고 POST /api/token/refresh 요청
- 서버는 refreshToken 이 올바른 토큰인지 검사 후 올바를 경우 accessToken, refreshToken 을 새로 발급
- 서버에서 응답헤더에 accessToken 을 담고, HTTPOnly Cookie 에 refreshToken 을 담아서 200 을 클라이언트로 응답 (accessToken, refreshToken 발급)



AccessToken, RefreshToken - refreshToken 만료시

refreshToken 만료시 재로그인 유도 절차

- 클라이언트의 특정 작업 수행 도중 refreshToken 만료 (서버에서 {X-Token-Relogin-Required: true} 응답헤더와 함께 401 UnAuthorized 를 응답)
- 클라이언트는 token 무효화 + /login 페이지로 리다이렉트



content-svc

- 소개

dailyfeed-content-svc 소개

- **Write / Delete Transaction 을 전담**
  - 주로 '기록' 또는 '삭제' 에 관련된 트랜잭션을 담당하며, '기록','삭제' 트랜잭션에 대한 스케일아웃 그룹입니다.
  - (참고) Read 에 관련된 트랜잭션들은 'timeline-svc' 에서 별도로 수행합니다.

- **본문 작성/수정/삭제, 댓글 작성/수정/삭제, 답글 작성/수정/삭제, 좋아요, 좋아요 취소**
  - dailyfeed-content-svc 의 기능은 본문 작성/수정/삭제, 댓글 작성/수정/삭제, 답글 작성/수정/삭제, 좋아요, 좋아요 취소 만 수행하는 간단한 구조입니다.

- **github**
  - https://github.com/alpha3002025/dailyfeed-content-svc

timeline-svc

- 소개

dailyfeed-timeline-svc 소개

- **Read Transaction 을 전담**
  - 주로 '조회'관련된 트랜잭션을 담당하며, '조회' 트랜잭션에 대한 스케일아웃 그룹입니다.
  - 슬로우 쿼리, 무거운 통계 쿼리 등이 발생하는 Read 트랜잭션은 Write 트랜잭션과 같은 인스턴스와 함께 수행되기 보다는 별도의 스케일아웃 그룹에서 실행되는게 효율적이라는 아이디어에서 구성한 서비스입니다.
  - (참고) Write/Delete 에 관련된 트랜잭션들은 'content-svc' 에서 별도로 수행합니다.

- **피드조회, 팔로잉 멤버의 최근 글, 댓글 많은 글, 인기 많은 글, 내가 쓴 글**
  - dailyfeed-timeline-svc 의 주요 기능은 '피드조회', '팔로잉 멤버의 최근 글', '댓글 많은 글', '인기 많은 글', '내가 쓴 글' 입니다.

- **github**
  - https://github.com/alpha3002025/dailyfeed-timeline-svc

image-svc

- 소개

dailyfeed-image-svc 소개

- **File IO : image 저장, 조회, 삭제 역할을 담당**
  - 주로 사용자의 썸네일 같은 image 에 대한 저장,조회,삭제 기능을 담당합니다.
  - S3 등을 도입할수도 있었겠지만, 트래픽 성격에 따른 스케일 아웃 그룹을 분류하는 예제를 보이기 위해 PVC 기반의 File IO 를 수행하는 인스턴스로 구성했습니다.

- **썸네일 조회**
  - 단건 조회 시에는 부하가 적지만, 피드 목록 조회 시에 함께 요청되는 프로필 이미지 등은 트래픽이 다수 발생합니다.

- **github**
  - https://github.com/alpha3002025/dailyfeed-image-svc

batch-svc

- 소개

dailyfeed-batch-svc 소개

- **batch 작업을 담당**
  - dead letter 처리, JWT 생성 서버 키 새로고침 등의 작업에 사용됩니다.

- **github**
  - https://github.com/alpha3002025/dailyfeed-batch-svc

installer

dailyfeed-installer

- 소개

dailyfeed-installer 소개

- **local infrastructure 설치, 애플리케이션 배포 (helm 기반)를 담당**
  - local,dev infrastructre 설치 코드들 입니다.
  - 혼자 개발하고, 아무에게도 공유할 필요가 없었다면 안만들었겠지만, 예제를 돌려봐야 하는 다른 사람의 입장도 생각해서 개설

- **주요 spec (local)**
  - kind 기반의 k8s 클러스터 구성, local kind 내에 istio 설치
  - mysql,redis,mongodb,kafka 의 경우 local pc 내에 docker-compose 를 통해 설치
  - docker-compose network 를 kind 클러스터에 공유해서 서로 통신이 가능하도록 구성
  - 설치 스크립트로 설치 가능
  - infrastructure, app-helm 으로 구성되어 있으며 infrastructure 는 mysql,redis,mongodb,kafka, k8s 클러스터 설치를 담당, app-helm 은 애플리케이션의 설치를 담당

- **github**
  - https://github.com/alpha3002025/dailyfeed-installer

infrastructure, CI/CD

dailyfeed-infrastructure

- 소개

kafka

dailyfeed-installer 구조

dailyfeed-installer

- **dailyfeed-infrastructure** — MySQL, MongoDB, Redis, Kafka 설치 스크립트
- **dailyfeed-app-helm** — backend, frontend 애플리케이션 helm chart, istio, hpa 설치 스크립트

dailyfeed-installer

e,g,

local profile : source local-install-infra-and-app.sh main

dev profile : source dev-install-infra-and-app.sh main

(dailyfeed-infrastructure)

cd dailyfeed-infrastructure

source install-local-hybrid.sh main

(1) docker-compose up -d

(2) docker network connect

(3) ingress nginx 설치, ingress-nginx controller 설치, ingress-nginx port 설정 수정

(4) dailyfeed namespace 생성 + istio-injection label 설정

(5) kind 리소스 설정 업그레이드 (resources requests,limit 수정)

(6) 주요 Configmap, Secret 설치 (kubectl apply)

(7) JWT 생성 비공개키 생성 (python)

(8) kind debuging 용도의 nodeport 들 생성

(9) Istio 설정 (PeerAuthentication, IngressGateway, External Service, ServiceEntry, DestinationRule)

(10) Kiali, Jaeger, Prometheus, Grafana 설치

(11) Kiali, Jaeger, Prometheus, Grafana 에 대한 NodePort 설치

(dailyfeed-app-helm)

cd dailyfeed-app-helm

source install-local.sh main

(member)

cd member

source install-helm-local.sh main

(content)

cd content

source install-helm-local.sh main

(timeline)

cd timeline

source install-helm-local.sh main

(search)

cd search

source install-helm-local.sh main

(image)

cd image

source install-helm-local.sh main

(activity)

cd activity

source install-helm-local.sh main

처음부터 다시 개발 한다면?

설치 코드 단순화, 불필요 인프라 제거

- kafka 를 로컬에 설치해서 진행했는데, 다시 새로운 프로젝트로 새로운 개발작업을 한다면, kafka 는 배제하고 feign 기반의 통신을 할 예정입니다. 1인이 운영하는 서비스를 목표로 하기에, 카프카 까지 운영하기에는 비용상으로 압박이 상당합니다. 애플리케이션의 구조 역시 단순하게 하고 싶기에 kafka 는 차기 개발 버전에서는 제외하고 애플리케이션의 설치 방식이 단순해지도록 구성할 예정입니다.

로컬 infra 제거

- 비용 이슈로 로컬 PC 에 대부분의 infrastructure를 설치하도록 구성했는데, 차기 버전을 새로 개발 시에는 redis,mysql,mongodb 등의 인프라는 모두 클라우드 서비스를 이용할 예정입니다.

feign, circuit breaker, rate limiter

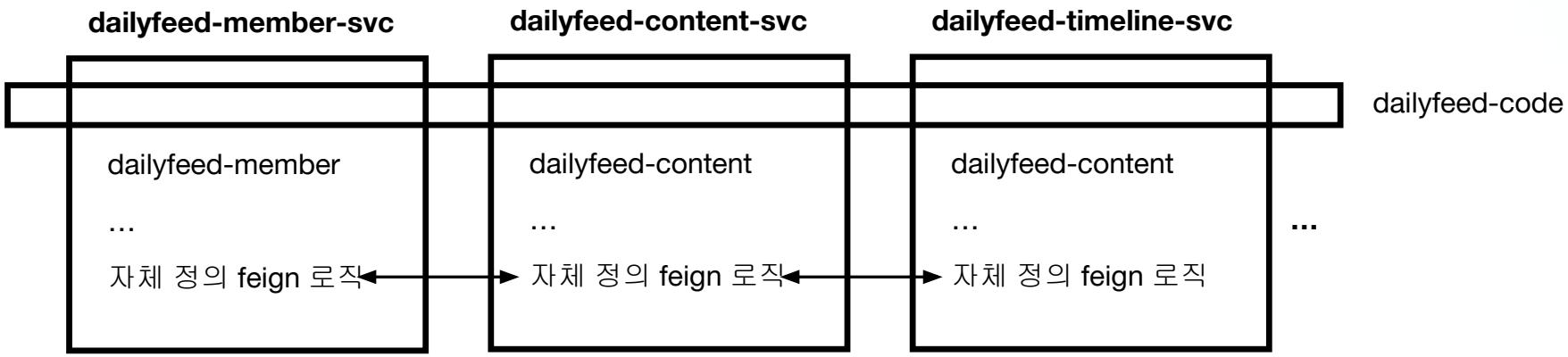
Resilience4j feign + circuit breaker, rate limiter

- dailyfeed-feign : feign 기능의 모듈화
- FeignHelper, FeignClient, FeignResponseHandler
- circuit breaker
- circuit breaker 설정의 강도 분류
- circuit breaker 설정 예시 (default)
- retry
- rate limiter

kafka

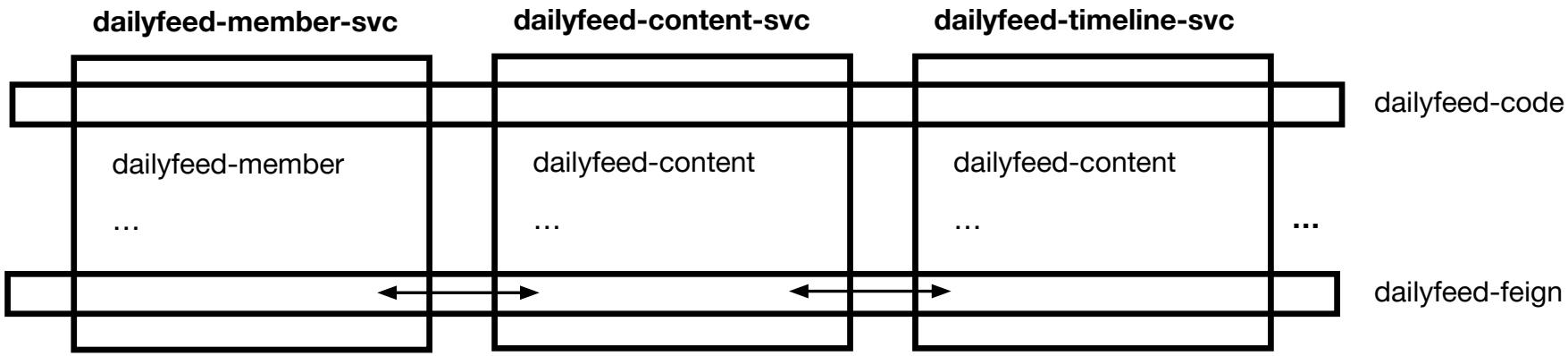
dailyfeed-feign : feign 기능의 모듈화

(before) feign 모듈화 적용 전



(after) feign 기능을 모듈화 (dailyfeed-feign)

feign 기능/엔드포인트의 코드를 모듈로 추출 및 정의 후 인터페이스 역할을 수행하도록 정의



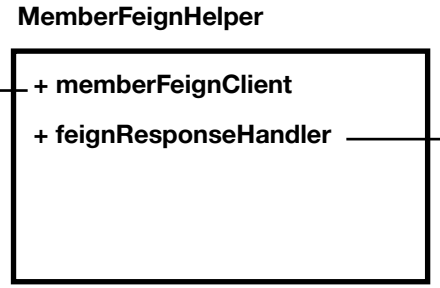
kafka

FeignHelper, FeignClient, FeignResponseHandler

OOOFeignClient : Feign 통신 전담

OOOFeignHelper : Feign 통신의 요청,응답을 FeignClient, FeignResponseHandler 객체를 이용해 관리 (Proxy 역할을 수행)

FeignResponseHandler : 응답 헤더 처리(e.g. 헤더 전파, 별도 처리)관련 코드 들을 모듈화



MemberFeignClient.java

```
e.g.
public interface MemberFeignClient {

    @RequestLine("GET /api/members”)
    @Headers("Authorization: Bearer {token}")
    Response getMember(@Param("token") String token);

    // …

}
```

FeignResponseHandler.java

```
e.g.
public class FeignResponseHandler {
    // …

    public void checkResponseHeadersAndStatusOrThrow(…) {

        // (1) 200,300 응답코드는 bypass
        // (2) 401,403,404,500 응답코드에 대한 별도 처리 (헤더 전파 or Exception
throw)

    }

    // …

}
```

kafka

CircuitBreaker

CircuitBreaker

장애 판정 : 현재 서비스에서 다른 서비스와의 API 호출 시 장애라고 판정할수 있는 기준을 정의 및 제공 가능

회복 판정 : 차단된 API 에 대해 회복되었다고 판정할 수 있는 기준을 정의 및 제공 가능

회로 차단 : 급작스러운 장애로 인한 서비스 다운시 현재 서비스에 장애가 전파되지 않도록 해당 서비스로의 호출을 차단하는 기능을 정의 및 제공 가능

Self Healing : 차단된 API 가 정상적으로 회복되었는지 주기적으로 체크해 회복으로 판정하고 CLOSED 상태로 전환할수 있는 기능 제공 (HALF_OPEN)

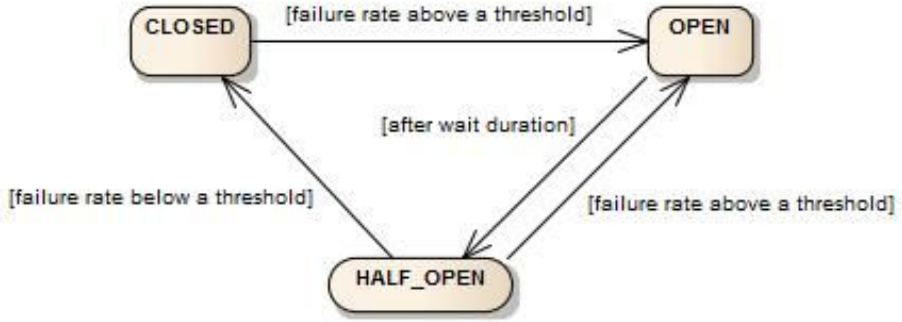
주요 상태

CLOSED : 닫힘 (정상)

OPEN : 열림 (차단)

HALF_OPEN : 반 열림 (테스트 상황)

- HALF_OPEN 시에 서비스 회복 여부를 체크하는 네트워크 테스트가 내부적으로 수행되며,
- 테스트 결과에 따라 CLOSED(정상복구), OPEN(차단)으로 변화



kafka

CircuitBreaker 설정의 강도 분류

Critical

- timeline-svc, member-svc
- 중요한 서비스로 분류하는 경우에 대해 ‘critical’ 로 정의
- 장애 판정을 엄격하게 정의

Default

- content-svc, activity-svc
- 가장 기본적인 경우, 일반적인 경우들에 대해 ‘default’ 로 정의

Bulk

- image-svc
- 프로필 이미지 조회 처럼 트래픽이 잦은 케이스

kafka

CircuitBreaker 설정 예시 (default)

max-attempts: 3
- 최대 3번까지 시도 (최초 1번 + 재시도 2번), 실패 시 총 3번의 호출 시도 후 최종 실패 처리

wait-duration: 500ms
- 각 재시도 사이의 대기 시간 (실패 후 재시도 하지않고 500ms 를 기다린 후 재시도)
- 장점
  - Backoff 부여
  - 일시적 장애 회복 시간 제공 : retry 하기 전에 term 을 두어서 회복시간 부여
  (참고) 대상 서비스에 즉시 재시도하지 않고 일정시간 대기하는 것을 Backoff 라고 합니다.

retry-exception:
- 어떤 exception 발생시 재시도 할지

ignore-exceptions:
- 어떤 exception 발생시 재시도를 하지 않을지

```yaml
retry:
  configs:
    # 기본 재시도 설정
    default:
      max-attempts: 3
      wait-duration: 500ms
      retry-exceptions:
        - feign.RetryableException
        - java.io.IOException
        - java.net.SocketTimeoutException
        - java.util.concurrent.TimeoutException
      ignore-exceptions:
        - feign.FeignException$BadRequest
        - feign.FeignException$Unauthorized
        - feign.FeignException$Forbidden
        - feign.FeignException$NotFound
```

kafka

CircuitBreaker 설정 예시 (default)

sliding-window-type: COUNT_BASED
- 슬라이딩 윈도우 타입 (COUNT_BASED(호출횟수 기반), TIME_BASED(시간 기반))

sliding-window-size: 100
- 슬라이딩 윈도우 크기. 최근 N개의 호출을 기준으로 실패율을 계산

minimum-number-of-calls: 10
- 최소 호출 횟수. 이 횟수 이상 호출되어야 circuit breaker 가 실패율을 계산

failure-rate-threshold: 50
- 실패율 임계값(%) : 이 비율 이상 실패시 Circuit OPEN

slow-call-rate-threshold: 80
- 느린 호출 기준 시간. 이 시간 이상 걸릴 경우 느린 호출로 간주

wait-duration-in-open-state: 60s
- OPEN 상태 대기 시간. Circuit 이 OPEN 된 후 HALF_OPEN 으로 전환되기까지의 대기 시간

permitted-number-of-calls-in-half-open-state: 10
- HALF_OPEN 상태에서 허용할 호출 수. 이 횟수 만큼 테스트 호출 후 CLOSED/OPEN 결정

automatic-transition-from-open-to-half-open-enable: true
- 자동 전환 활성화 : OPEN → HALF_OPEN 자동 전환 여부

```yaml
# Resilience4j 설정
resilience4j:
  circuitbreaker:
    configs:
      # 기본 설정 (일반적인 서비스)
      default:
        # 슬라이딩 윈도우 타입: COUNT_BASED(호출 횟수 기반) 또는 TIME_BASED(시간 기반)
        sliding-window-type: COUNT_BASED
        # 슬라이딩 윈도우 크기: 최근 N개의 호출을 기준으로 실패율 계산
        sliding-window-size: 100
        # 최소 호출 횟수: 이 횟수 이상 호출되어야 Circuit Breaker가 실패율을 계산
        minimum-number-of-calls: 10
        # 실패율 임계값(%): 이 비율 이상 실패 시 Circuit OPEN
        failure-rate-threshold: 50
        # 느린 호출 비율 임계값(%): 이 비율 이상이 느린 호출이면 Circuit OPEN
        slow-call-rate-threshold: 80
        # 느린 호출 기준 시간: 이 시간 이상 걸리면 느린 호출로 간주
        slow-call-duration-threshold: 5s
        # OPEN 상태 대기 시간: Circuit이 OPEN된 후 HALF_OPEN으로 전환되기까지 대기 시간
        wait-duration-in-open-state: 60s
        # HALF_OPEN 상태에서 허용할 호출 수: 이 횟수만큼 테스트 호출 후 CLOSED/OPEN 결정
        permitted-number-of-calls-in-half-open-state: 10
        # 자동 전환 활성화: OPEN → HALF_OPEN 자동 전환 여부
        automatic-transition-from-open-to-half-open-enabled: true

      # 중요 서비스용 (Timeline, Member 등 사용자 UX에 직접 영향)
      critical:
        sliding-window-type: COUNT_BASED
        sliding-window-size: 50        # 빠른 감지를 위해 작은 윈도우 사용
        minimum-number-of-calls: 5     # 5번만 호출되어도 패턴 감지
        failure-rate-threshold: 40     # 더 민감한 임계값 (40% 실패 시 OPEN)
        slow-call-rate-threshold: 70   # 느린 호출에 대해서도 민감하게 반응
        slow-call-duration-threshold: 3s  # 3초 이상이면 느린 호출
        wait-duration-in-open-state: 30s  # 빠른 복구 시도 (30초)
        permitted-number-of-calls-in-half-open-state: 5  # 적은 테스트 호출로 빠른 결정
        automatic-transition-from-open-to-half-open-enabled: true
```

kafka

retry

retry 란?

현재 서비스에서 다른 서비스와의 API 호출 시 에러 발생시 재시도 횟수, 재시도 전 timeout 정도, 어떤 Exception 에 대한 재시도를 할지 등을 정의

장점

- 불필요한 재시도 방지 (e.g. 4xx 등에 대해서는 즉시 실패 처리)
- 의미있는 재시도 (e.g. 네트워크 장애, 타임아웃의 경우 재시도 하도록 설정)
- Backoff 전략으로 부하 분산 (Backoff 전략: 시스템이 실패 후 재시도할 때 재시도하지 않고 일정시간 대기하는 전략)

Backoff 의 종류

Fixed Backoff (고정 백오프) : 매번 동일한 시간 대기 (e.g. 500ms → 500ms → 500ms)

Exponential Backoff (지수 백오프) : 재시도마다 대기시간이 지수적으로 증가 (e.g. 100ms → 200ms → 400ms → 800ms)

Linear Backoff (선형 백오프) : 재시도마다 대기시간이 선형적으로 증가 (e.g. 100ms → 200ms → 300ms → 400ms)

kafka

retry 설정의 강도 분류

Critical

- timeline-svc, member-svc
- 중요한 서비스로 분류하는 경우에 대해 ‘critical’ 로 정의
- 장애 판정을 엄격하게 정의

Default

- content-svc, activity-svc
- 가장 기본적인 경우, 일반적인 경우들에 대해 ‘default’ 로 정의

Conservative

- image-svc
- 프로필 이미지 조회 처럼 트래픽이 잦은 케이스는 예외적인 경우로 분류

kafka

retry 설정 예시 (default)

max-attempts: 3
- 최대 3번까지 시도 (최초 1번 + 재시도 2번), 실패 시 총 3번의 호출 시도 후 최종 실패 처리

wait-duration: 500ms
- 각 재시도 사이의 대기 시간 (실패 후 재시도 하지않고 500ms 를 기다린 후 재시도)
- 장점
  - Backoff : 대상 서비스에 즉시 재시도하지 않고 일정시간 대기하는 것을 Backoff 라고 합니다.
  - 일시적 장애 회복 시간 제공 : retry 하기 전에 term 을 두어서 회복시간 부여

retry-exception:
- 어떤 exception 발생시 재시도 할지

ignore-exceptions:
- 어떤 exception 발생시 재시도를 하지 않을지

```yaml
retry:
  configs:
    # 기본 재시도 설정
    default:
      max-attempts: 3
      wait-duration: 500ms
      retry-exceptions:
        - feign.RetryableException
        - java.io.IOException
        - java.net.SocketTimeoutException
        - java.util.concurrent.TimeoutException
      ignore-exceptions:
        - feign.FeignException$BadRequest
        - feign.FeignException$Unauthorized
        - feign.FeignException$Forbidden
        - feign.FeignException$NotFound
```

kafka

rate limiter

rate limiter 란?

현재 서비스에서 다른 서비스와의 API 호출 시 outbound 요청의 빈도(rate)를 제한하는 기능

장점

- self-protection : 다른 서비스에 과부하를 주지 않도록 보호해줄 수 있다.
- 연쇄 장애 방지 : 현재 서비스에서의 과도한 요청으로 인해 전체 시스템에 영향을 주지 않도록 함
- 호출 주기 안정화 : 특정 서비스로의 급격한 트래픽 증가시 제한을 걸수 있다.
  - 일정 사이즈의 최소/최대 호출 가능 rate 를 지정해서 현재 시스템의 outbound 트래픽의 허용치를 규격화 가능

kafka

rate limiter 설정의 강도 분류

Critical

- timeline-svc, member-svc
- 중요한 서비스로 분류하는 경우에 대해 ‘critical’ 로 정의
- 장애 판정을 엄격하게 정의

Default

- content-svc, activity-svc
- 가장 기본적인 경우, 일반적인 경우들에 대해 ‘default’ 로 정의

Conservative

- image-svc
- 프로필 이미지 조회 처럼 트래픽이 잦은 케이스는 예외적으로 분류

kafka

rate limiter 설정 예시 (default)

limit-refresh-period : 1s
- 제한 갱신 주기
- 1초로 지정했습니다.
- 1초마다 기록했던 rate 가 리셋됩니다.

limit-for-period : 100
- 갱신 주기 당 허용 호출 수
- limit-refresh-period 구간 내에서 '허용할 호출 수' 를 의미합니다.
- 1초에 100번 호출 가능하다는 것을 의미합니다.

timeout-duration : 0s
- 허용량 초과시 대기시간
- 0s 는 대기하지 않고 즉시 실패하는 것을 의미합니다.

```yaml
ratelimiter:
  configs:
    # 기본 Rate Limiter 설정
    default:
      # 제한 갱신 주기: 1초마다 제한이 리셋됨
      limit-refresh-period: 1s
      # 갱신 주기당 허용 호출 수: 1초에 100번 호출 가능
      limit-for-period: 100
      # 허용량 초과 시 대기 시간: 대기하지 않고 즉시 실패
      timeout-duration: 0s

    # 중요 서비스용 (더 많은 요청 허용)
    critical:
      limit-refresh-period: 1s
      limit-for-period: 200  # 1초에 200번 호출 가능 (부하가 높을 수 있음)
      timeout-duration: 100ms  # 100ms까지 대기 후 실패

    # 보수적 Rate Limiter (외부 서비스 보호)
    conservative:
      limit-refresh-period: 1s
      limit-for-period: 50  # 1초에 50번으로 제한 (외부 서비스 부담 감소)
      timeout-duration: 0s

  instances:
    timelineService:
      base-config: critical  # Timeline은 트래픽이 많으므로 여유있게
    memberService:
      base-config: critical
    contentService:
      base-config: default
    activityService:
      base-config: default
    imageService:
      base-config: conservative  # Image 서비스는 부담이 크므로 제한
```

Kafka 사용 예제

dailyfeed-activity-svc

- dead letter 처리
- Acks=1, At Least Once 처리 구조
- 중복 메시지 체크 방식
- 날짜별 토픽 ({topicName}-yyyyMMdd})
- feign 으로 대체할 경우

kafka

dailyfeed-activity-svc 소개

- kafka 사용시 어떻게 사용할지에 대한 예시를 위한 project
  - kafka 사용 시 설계 아이디어를 설명하기 위한 쇼케이스 project
  - frontend 에는 해당 기능 배제 (설계 개념만을 설명하기 위한 예제)

- 사용자가 글 생성/수정/삭제, 댓글 생성/수정/삭제 작업을 한 활동 로그를 기록하는 project

- Publish : dailyfeed-content-svc → kafka topic
- Listen : kafka-topic → dailyfeed-activity-svc
- Listen 시 장애 처리
  - kafka_listener_dead_letters 컬렉션에 해당 메시지 insert → dailyfeed-batch-svc 에서 KafkaListener 에서 실패한 데이터 처리
  - kafka_listener_dead_letters 컬렉션에 데이터 저장 실패 시 → kafka consumer offset commit 보류 (rollback)
- 사용자의 글 생성/수정/삭제, 댓글 생성/수정/삭제
  - 글 생성/수정/삭제 : POST_CREATE, POST_UPDATE, POST_DELETE 메시지 발송 → dailyfeed-activity-svc 에서 처리
  - 댓글 생성/수정/삭제 : COMMENT_CREATE, COMMENT_UPDATE, COMMENTT_DELETE 메시지 발송 → dailyfeed-activity-svc 에서 처리

kafka

Kafka 통신 시 데드레터 처리

Publisher Error

발행 실패 시 **kafka_publisher_dead_letters** (MongoDB)에 저장 후 배치를 통해 재발송 수행.

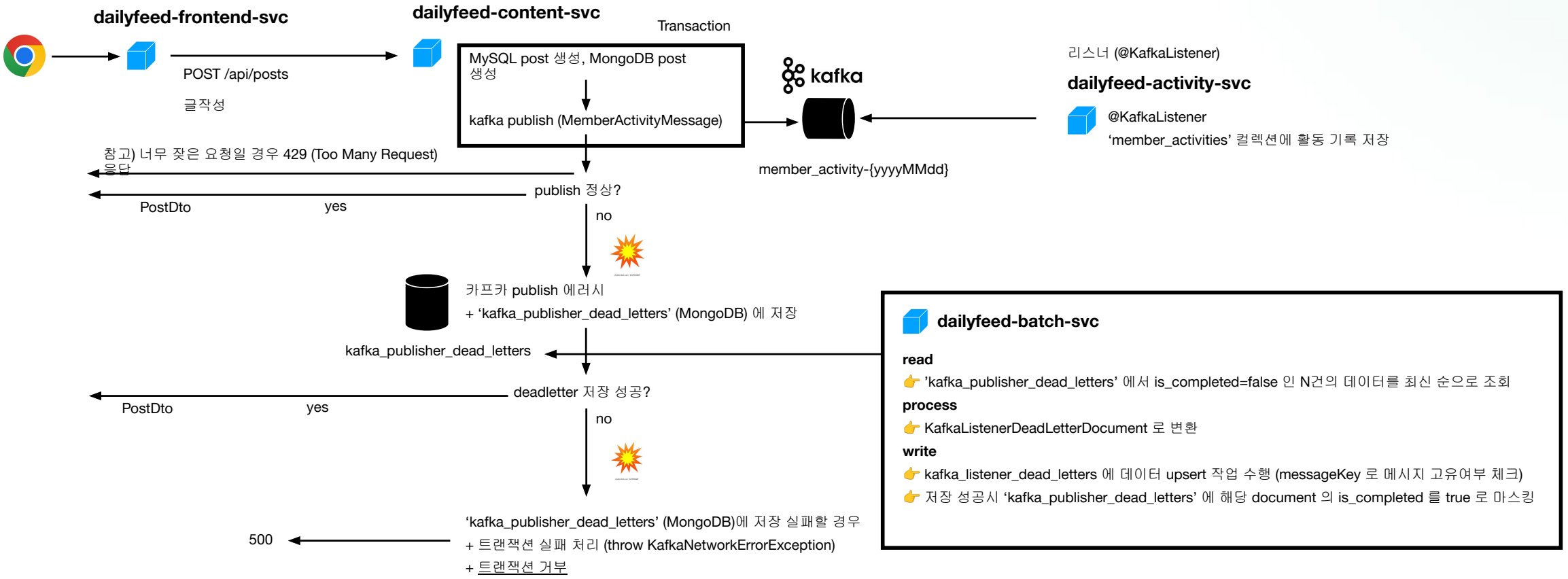
사용자의 글 쓰기/삭제/수정, 댓글 쓰기/삭제/수정 요청시
- dailyfeed-content-svc 에서 트랜잭션의 마지막에 kafka 로 kafka publish 시도
- kafka publish 에 실패하면 ? : kafka_publisher_dead_letters 에 해당 메시지 저장
- kafka_publisher_dead_letters 에 메시지 저장 실패하면? : 500 Internal Server Error 응답과 함께 트랜잭션 거부

Listener Error

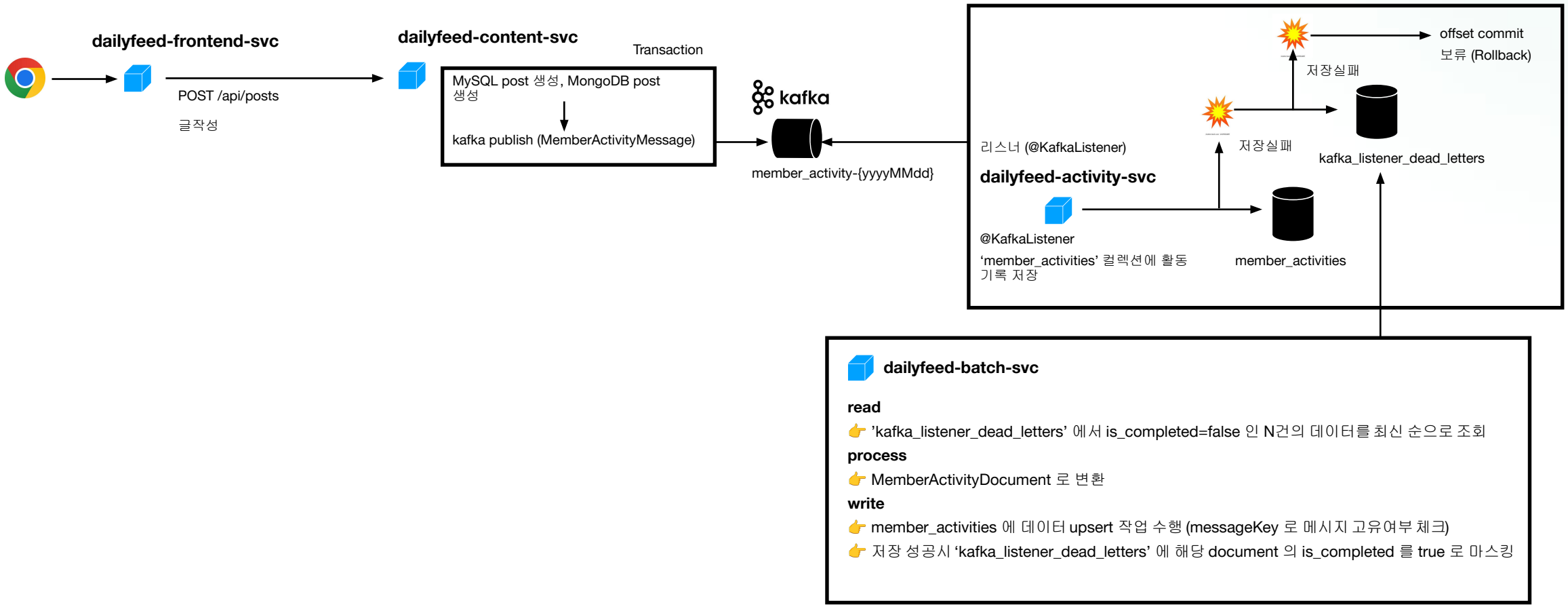
수신 실패 시 **kafka_listener_dead_letters**에 로깅 후 보정 작업을 통해 메시지 유실 방지.

사용자의 글 쓰기/삭제/수정, 댓글 쓰기/삭제/수정 활동 기록
- member_activities 컬렉션 저장 실패하면 ? : kafka_listener_dead_letters 에 해당 메시지 저장
- kafka_listener_dead_letters 에 메시지 저장 실패하면? : offset commit 보류 (rollback)

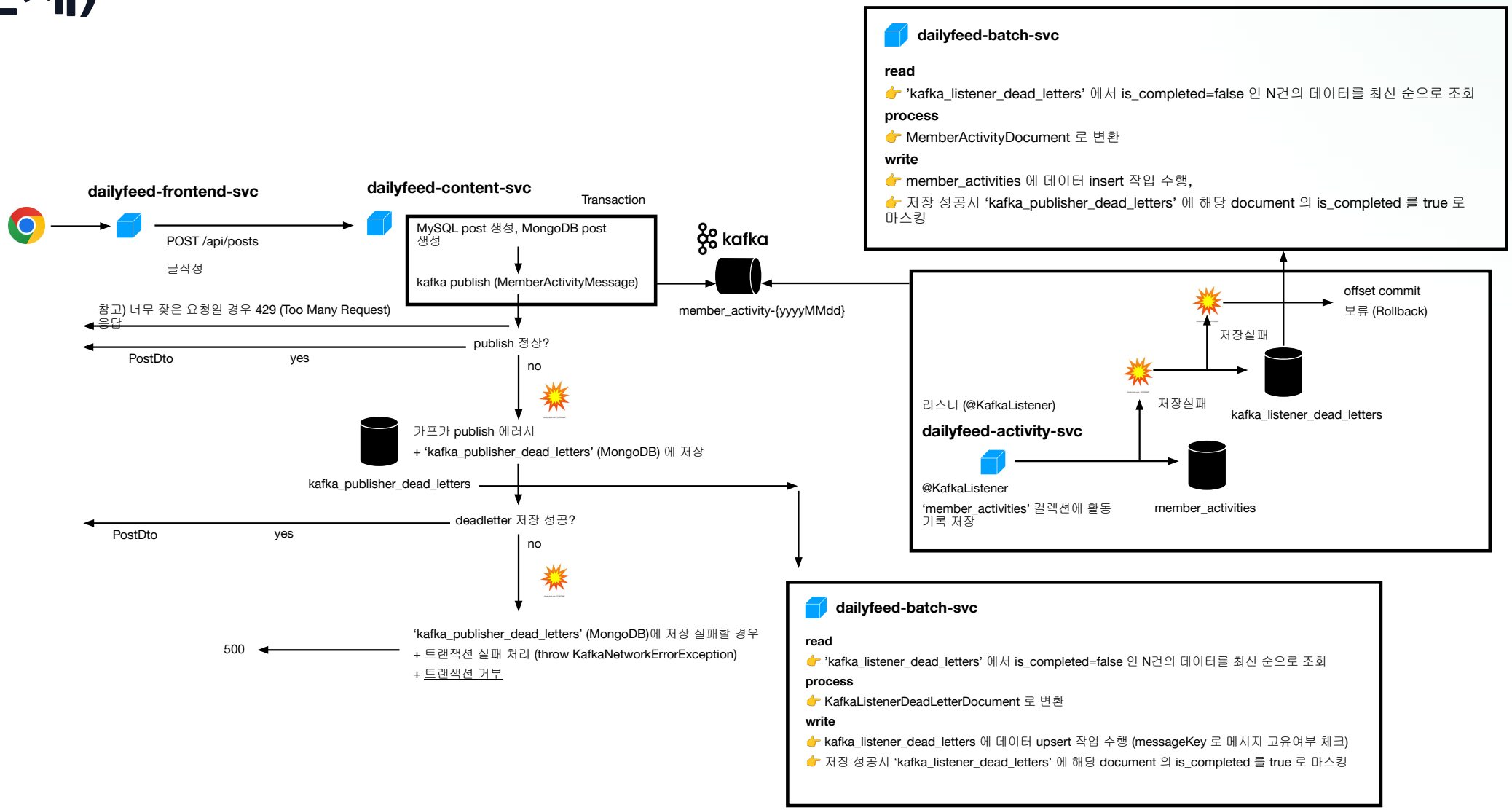
Kafka publisher 통신 에러 발생 시 데드레터 처리



Kafka listener 통신 에러 발생 시 데드레터 처리



Kafka publisher, listener 통신 에러 발생 시 데드레터 처리 (전체)



kafka

Acks=1, At Least Once 처리 구조

Producer Acknowledgement : Acks = 1 선택

- 리소스/비용을 줄이기 위해 리더 파티션 1기만 운영하는 경우도 함께 고려

Consumer Offset : At Least Once 선택

- 메시지 수신 시 중복 체크 : 중복된 메시지를 받더라도 Redis 를 통해 중복 메시지를 체크
- 데이터 저장 시 중복 처리 : 데이터 저장 시 같은 메시지가 저장하게될 경우 역시 고려해 upsert 수행

참고) Producer Acknowledgement

acks = 1

- Producer 가 메시지를 브로커에 보낼때 리더파티션에 메시지를 보낸 후, 리더파티션이 메시지를 받아서 로그에 쓴 후 리더로부터 ack 을 받는 방식입니다.

acks = 0

- Producer 가 메시지를 브로커에 보낼때 메시지를 보내기만 하고 ack 를 기다리지 않는 방식입니다.

acks = all (-1)

- Producer 가 메시지를 브로커에 보낼때 메시지를 보낸 후 리더파티션 & 모든 ISR 이 메시지를 받은 후 ack 을 받는 방식입니다. (복제본에 모두 복제되야 acks 를 받음)

acks = all 을 사용할 경우 min.insync.replicas=2 등으로 최소 한도의 복제 본 수를 지정해야 합니다.

현재 프로젝트에서는 일반적으로 많이 설정되는 Acks = 1 로 지정했습니다. Acks = 1 로 지정하면 리더가 죽고 팔로워가 복제를 받지 못한 경우 데이터가 손실될 가능성이 있지만 개발 버전의 환경상 많은 리소스가 불필요하고, 리더파티션 1기만 운영할 경우도 있기에 acks = 1 로 지정했습니다. 만약 acks=all 을 선택할 경우는 꼭 'min.insync.replcas' 를 지정해야 합니다.

참고) Consumer Offset

At Most Once (최대 한번)

- Offset 을 먼저 커밋하고, 나중에 처리하는 방식입니다. 처리 중 실패할 경우 메시지가 손실됩니다.
- 최대 1번 처리됩니다.
- 메시지의 중복처리는 없지만, 데이터 손실가능성이 존재합니다.

At Least Once (최소 한번)

- 처리를 먼저 하고 Offset을 나중에 커밋하는 방식입니다. 처리 후 커밋 전 실패할 경우 재처리 됩니다. (커밋이 안된 메시지는 재수신)
- 데이터의 손실은 없지만 메시지가 중복 수신 됩니다. 따라서 애플리케이션 레벨에서 메시지 중복 시에 대한 처리를 해주면 메시지 유실 없이 카프카 통신이 가능합니다.
- 가장 일반적으로 사용되는 방식입니다.

Exactly Once (정확히 한번)

- 처리와 커밋이 하나의 트랜잭션으로 묶입니다. 둘 다 성공하거나 둘 다 실패합니다. 정확히 1번 처리됩니다.
- 중복처리가 없고 데이터 손실 역시 없는 방식이지만 성능 오버헤드가 있으며 구현이 복잡하고 추가설정이 필요합니다.

kafka

Consumer Offset Commit 방식을 At Least Once 를 채택했기에 같은 메시지가 여러 번 수신될 수 있기에 여기에 대한 처리 방식을 정리합니다.

중복 메시지 체크 방식

메시지 중복체크

메시지 전송 시 메시지 키를 발급, 메시지 수신 시에 Redis/Database 에서 중복 수신 여부를 체크하는 방식을 사용

메시지 키 형식

POST_CREATE

"member_activity:kafka_event:POST_CREATE###{postId}###{memberId}"

POST_UPDATE,POST_DELETE,POST_READ

"member_activity:kafka_event:{POST_UPDATE|POST_DELETE|POST_READ}###{postId}###{memberId}###{yyyy-MM-dd HH:mm:ss.SSSSSSSSS}"

COMMENT_CREATE

"member_activity:kafka_event:COMMENT_CREATE###{postId}###{memberId}"

COMMENT_UPDATE,COMMENT_DELETE,COMMENT_READ

"member_activity:kafka_event:{COMMENT_UPDATE|COMMENT_DELETE|COMMENT_READ}###{commenttId}###{memberId}###{yyyy-MM-dd HH:mm:ss.SSSSSSSSS}"

LIKE_POST, LIKE_POST_CANCEL

"member_activity:kafka_event:{LIKE_POST|LIKE_POST_CANCEL}###{postId}###{memberId}"

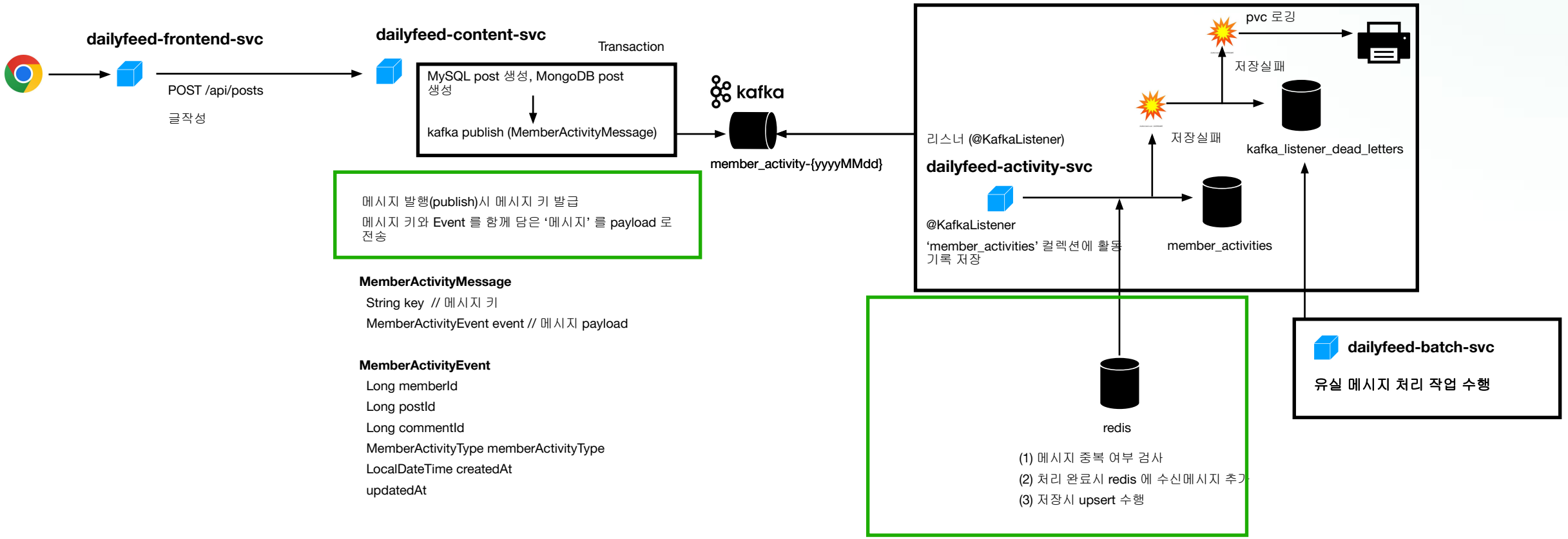
LIKE_COMMENT, LIKE_COMMENT_CANCEL

"member_activity:kafka_event:{LIKE_COMMENT|LIKE_COMMENT_CANCEL}###{commentId}###{memberId}"

kafka

Consumer Offset Commit 방식을 At Least Once 를 채택했기에 같은 메시지가 여러 번 수신될 수 있기에 여기에 대한 처리 방식을 정리합니다.

중복 메시지 체크 방식



kafka

날짜 별 토픽 ({topicName}-yyyyMMdd)

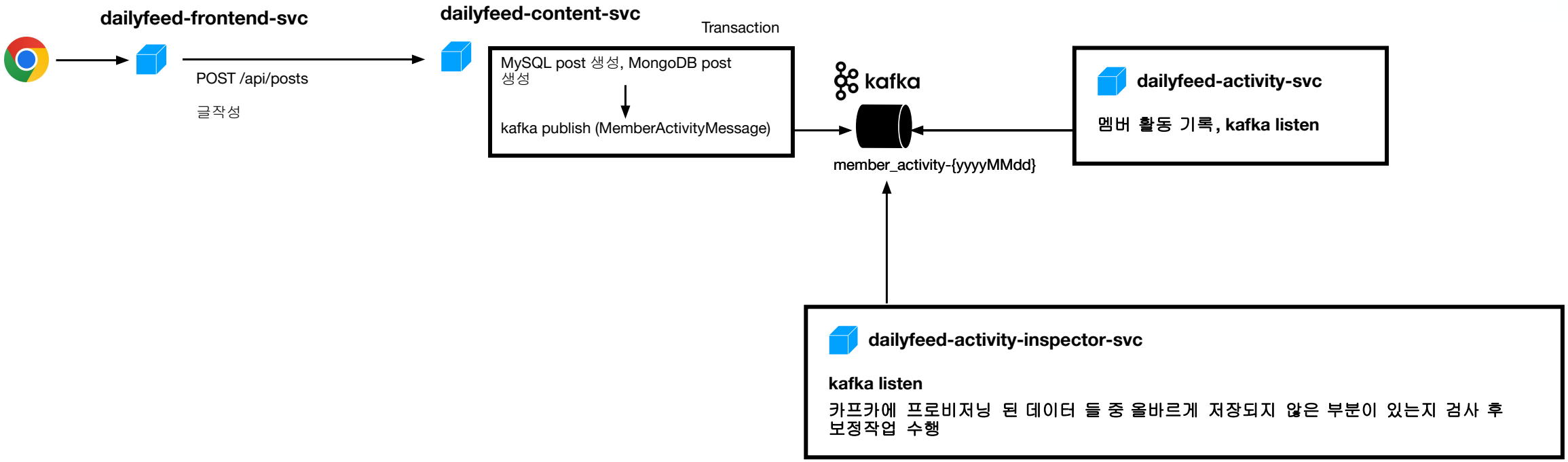
날짜 별 토픽을 도입하게 된 이유

후보정 작업 가능

- 데이터의 오류가 있는지 등에 대한 후보정 작업에 대해 유연하게 전략을 취할수 있다는 점

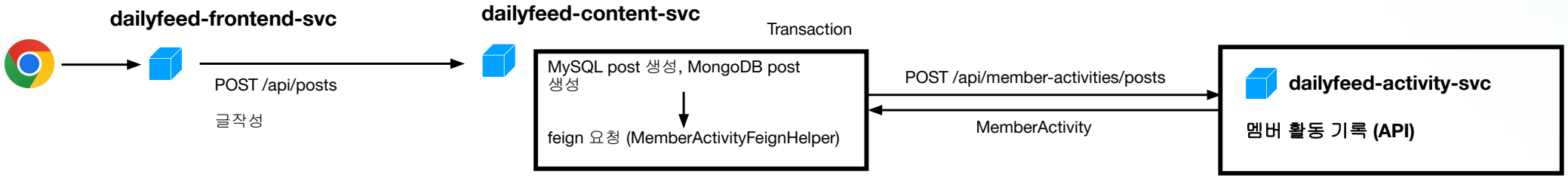
운영비용 최적화

- 이미 처리 완료된 데이터에 대한 토픽(e.g. 7일전 날짜의 토픽)의 경우 주기적으로 토픽 삭제를 통해 디스크 사이즈를 줄일 수 있다는 점

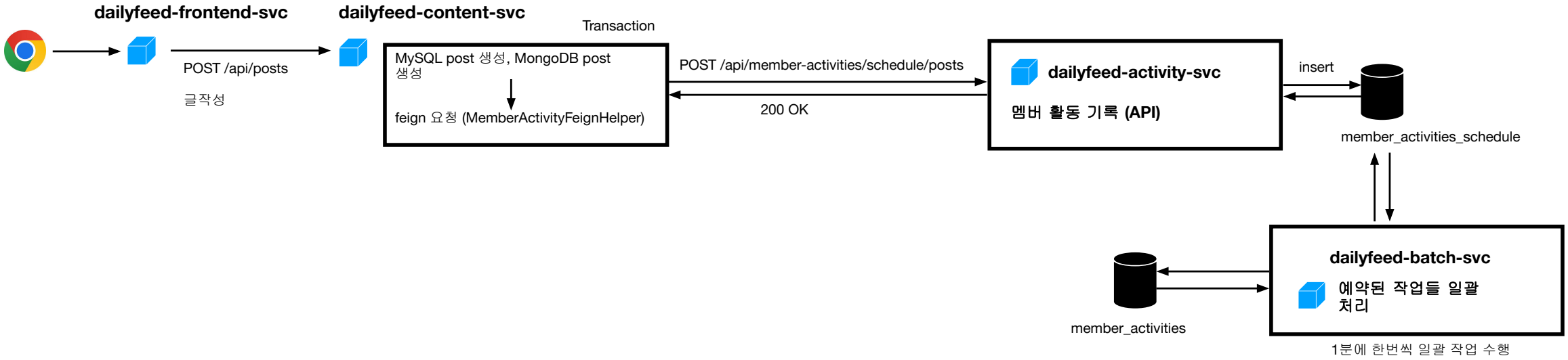


feign 으로 대체할 경우

동기적으로 구성할 경우 ('dailyfeed' 프로젝트의 현재 구성)



비동기적으로 구성할 경우



coding convention

Coding Convention

- common 모듈 패턴 지양 → 구체적인 sub module 로 분리개발
- dailyfeed-code : 서비스, 모듈 간 interface 역할
- -Mapper : 객체 생성/변환 컴포넌트
- get-OrThrow
- boolean 대신 Predicate 역할의 Enum 사용 (80%)

common 모듈 패턴 지양 → 구체적인 sub module 로 분리개발

common 모듈 패턴 지양, sub module 로 구체적인 모듈 개발

- 현재 프로젝트에서 common 모듈은 완전히 사용하지 않습니다.
- common 모듈 대신 submodule 로 필요한 기능들을 구체적인 모듈로 정의해서 사용했습니다.
- e.g. dailyfeed-code, dailyfeed-redis-support, dailyfeed-kafka-support, dailyfeed-pagination-support, dailyfeed-deadletter-support
- 성능 측면 : common 모듈 사용시 common 에서 사용하는 스레드 풀, 메모리 사용량으로 인해 신규 서비스를 개발하더라도 common 모듈 사용시 고혈압,당뇨 상태의 인스턴스 양산되는 현상 자제
- 단점 : 해야하는 작업들이 많아짐
- 개발의 편리함은 결국 나중으로 갈 수록 관리의 불안함으로 발전하게 된다는 점을 고려, “지나친 공통화에 대한 욕심” 은 결국 “어리석은 똑똑함"

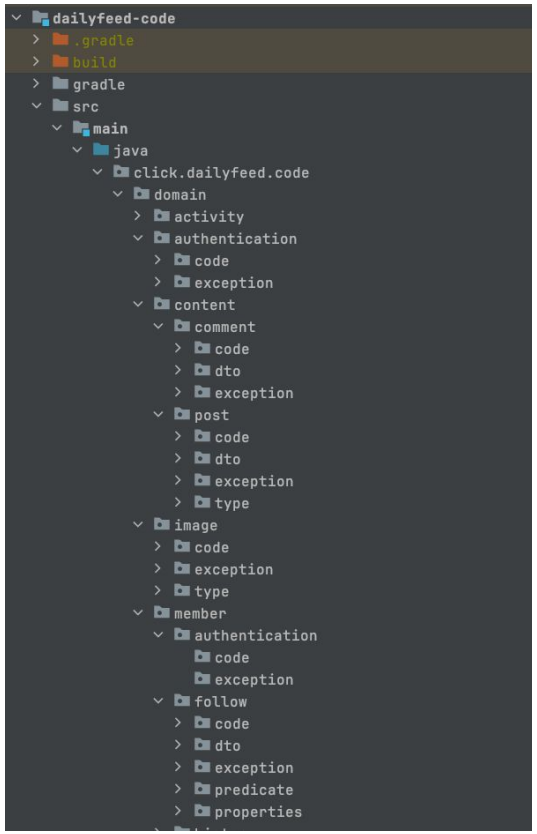
data 모듈 구성 자제

- 쿼리(JPA, Querydsl)를 별도의 특정 모듈에 모아두는 경우도 있는데 dailyfeed 에서는 이런 패턴을 사용하지 않았습니다.
- 구현을 편하게 하고 빠르게 하려면 쿼리(JPA, SQL, Querydsl)를 한 곳에 모아둘수 있었겠지만 서비스 별로 독립성을 부여해주기 위해 구체 쿼리 들을 개별 서비스에 보유하도록 구성했습니다.
- 예를 들어 dailyfeed-content-svc, dailyfeed-timeline-svc 에서 Post, Comment 라는 도메인은 각 서비스별로 서로 다른 쿼리(JPA, SQL, Querydsl)로 사용합니다.
- 즉, dailyfeed-content-svc 입장에서는 Post,Comment 의 작업은 독립성을 가지고 쿼리를 구성하고, dailyfeed-timeline-svc 입장에서도 Post,Comment 의 작업을 독립성을 가지고 쿼리를 구성하도록 했습니다.

dailyfeed-code : 서비스, 모듈 간 interface 역할

Dto, Exception, Enum

- 서비스간 API 를 통해 통신을 할때 사용되는 Dto, Exception, Enum 은 dailyfeed-code 라는 서브모듈에 정의했습니다.
- dailyfeed-code 는 common 모듈처럼 동작하는 모듈이 아닙니다.
- dailyfeed-code 에는 특정 메서드 나 비즈니스로직을 포함하지 않고, 상수, Enum, 예외코드, Exception, 데이터 표현 객체(Dto) 만을 담는 기능이 되도록 구성했습니다.
- 즉 dailyfeed-code 는 **어떤 값이나 상수, 예외코드, Exception, 데이터 표현 객체(Dto)만을 담고 있으며, 서비스간 통신 시 또는 모듈간 기능 혼합시 "인터페이스" 역할**을 하게 됩니다.



-Mapper : 객체 생성/변환 컴포넌트

-Mapper 컴포넌트

- 객체를 생성하거나 변환하는 로직들은 -Mapper 라는 이름을 가진 컴포넌트로 분류해서 정의했으며, @Component 로 선언해서 객체의 생명주기/의존성 주입을 Spring Framework 레벨에서 관리하도록 지정
- 객체 생성/변환을 하드 코딩으로 생성할 경우 테스트가 어려워진다는 단점, 객체 생성 로직이 이곳 저곳에 하드 코딩으로 존재하게 되어 유지보수가 편리하지 않다는 단점
- 객체 생성/변환을 별도의 컴포넌트에서 수행하도록 정의할 경우 개발 시에는 불편했지만 코드 수정작업 시에는 관리가 더 쉬워졌습니다.
- 전체 코드 중 98 % 정도가 -Mapper 컴포넌트를 사용하고 2% 정도는 아직 하드코딩된 객체 생성/변환 코드로 남아있습니다.

e.g. dailyfeed-member / MemberProfileMapper

```
@Mapper(componentModel = "spring", unmappedTargetPolicy = ReportingPolicy.IGNORE)  13 usages  1 implementation  alpha3002025
public interface MemberProfileMapper {
    MemberProfileMapper INSTANCE = Mappers.getMapper(MemberProfileMapper.class);  no usages

    default MemberProfileDto.MemberProfile fromEntity(MemberProfile memberProfile, Long followersCount, Long followingsCount) {
        return MemberProfileDto.MemberProfile.builder()
                .id(memberProfile.getId())
                .memberId(memberProfile.getMember().getId())
                .memberName(memberProfile.getMemberName())
                .handle(memberProfile.getHandle())
                .displayName(memberProfile.getDisplayName())
                .bio(memberProfile.getBio())
                .location(memberProfile.getLocation())
                .websiteUrl(memberProfile.getWebsiteUrl())
                .birthDate(memberProfile.getBirthDate())
                .gender(memberProfile.getGender())
                .languageCode(memberProfile.getLanguageCode())
                .countryCode(memberProfile.getCountryCode())
                .verificationStatus(memberProfile.getVerificationStatus())
                .privacyLevel((memberProfile.getPrivacyLevel() == null) ? null : memberProfile.getPrivacyLevel())
                .profileCompletionScore(memberProfile.getProfileCompletionScore())
                .isActive(memberProfile.getIsActive())
                .avatarUrl(memberProfile.getAvatarUrl())
                .coverUrl(memberProfile.getCoverUrl())
                .createdAt(memberProfile.getCreatedAt())
                .updatedAt(memberProfile.getUpdatedAt())
                .followersCount(followersCount)
                .followingsCount(followingsCount)
                .build();
    }
```

e.g. dailyfeed-content / CommentMapper

```
@Component  2 usages  alpha3002025 +1
public class CommentMapper {
    public CommentDto.Comment fromCommentNonRecursive(Comment comment, MemberProfileDto.Summary author){  3 usages
        return CommentDto.Comment.builder()
                .id(comment.getId())
                .content(comment.getContent())
                .authorId(comment.getAuthorId())
                .authorName(author != null ? author.getDisplayName() : MessageProperties.KO.DELETED_USER)
                .authorHandle(author != null ? author.getMemberHandle() : MessageProperties.KO.DELETED_HANDLE)
                .authorAvatarUrl(author != null ? author.getAvatarUrl() : MessageProperties.KO.NO_AVATAR_URL)
                .postId(comment.getPost().getId())
                .parentId(comment.getParent() != null ? comment.getParent().getId() : null)
                .depth(comment.getDepth())
                .createdAt(comment.getCreatedAt())
                .updatedAt(comment.getUpdatedAt())
                .build();
    }
```

get-OrThrow()

getOrThrow()

- 예외를 던지는 함수에는 가급적 '-OrThrow()' 접미사가 붙는 네이밍 패턴을 적용
- 트랜잭션 처리 로직, 인증 로직 등에 특정 함수가 예외를 던지는지 아닌지를 메서드 명으로 확실하게 알 수 있다면 해당 코드의 정의부까지 일일이 찾아서 확인하지 않아도 된다는 장점
- 현재 dailyfeed 백엔드 프로젝트 들 에서는 80% 정도의 로직들에 get-OrThrow() 네이밍 패턴을 사용 중

e.g. dailyfeed-feign / FeignResponseHandler

```java
public void checkResponseHeadersAndStatusOrThrow(Response feignResponse, HttpServletResponse httpResponse){
    final int status = feignResponse.status();
    if (status >= 200 && status < 300) {
        return;
    }

    if (status >= 400 && status < 500) {
        // HTTP 상태 코드에 따른 적절한 예외 처리
        if (status == 401) {
            log.error("Unauthorized request to member service - invalid or expired token");
            propagateTokenRefreshHeader(feignResponse, httpResponse);
            throw new MemberUnauthorizedException();
        } else if (status == 403) {
            log.error("Forbidden request to member service - insufficient permissions");
            throw new MemberForbiddenException();
        } else if (status == 404) {
            log.error("Member not found in member service");
            throw new MemberNotFoundException();
        } else if (status >= 500) {
            log.error("Member service internal error - status: {}", status);
            throw new MemberApiConnectionErrorException();
        } else {
            log.error("Unexpected member service error - status: {}", status);
            throw new MemberApiConnectionErrorException();
        }
    }
}
```

e.g. dailyfeed-member / AuthenticationService

```java
@Transactional(readOnly = true) 1 usage  alpha3002025
public Member getMemberOrThrow(AuthenticationDto.LoginRequest loginRequest) {
    return memberRepository
            .findFirstByEmailFetchJoin(loginRequest.getEmail())
            .orElseThrow(() -> new MemberNotFoundException());
}

public void checkIfPasswordMatchesOrThrow(String requestPassword, String encryptedPassword) {
    if (!passwordEncoder.matches(requestPassword, encryptedPassword)) {
        throw new MemberPasswordInvalidException();
    }
}
```

boolean 대신 Predicate 역할의 Enum 사용 (80%)

boolean return 타입사용은 최대한 자제 → 가급적 Predicate 역할의 Enum 사용

- 주로 인증로직에서 어떤 로직이 true/false 를 return 하더라도 Expired 됐다는 건지 NotExpired 라는 건지 의미가 모호해지는 케이스가 자주 있습니다.
- 현재 프로젝트에서는 70% 정도의 코드 들에 boolean return 대신 Predicate 역할의 enum 을 return 하도록 구성해둔 상태입니다. (간단한 boolean return 구문이고, 의미가 명확할 경우는 미적용)

e.g. dailyfeed-member / AuthenticationService

```
public DailyfeedServerResponse<MemberDto.Member> signup(AuthenticationDto.SignupRequest signupRequest) {
    if (MemberExistsPredicate.EXISTS.equals(checkIfMemberAlreadyExists(signupRequest))) {
        throw new MemberAlreadyExistsException();
    }

    Member newMember = authenticationMapper.newMember(signupRequest, passwordEncoder, roles: "MEMBER");
    Member saved = memberRepository.save(newMember);

    return DailyfeedServerResponse.<~>builder()
            .status(HttpStatus.CREATED.value())
            .result(ResponseSuccessCode.SUCCESS)
            .data(authenticationMapper.fromMemberEntityToMemberDto(saved))
            .build();
}
```

e.g. dailyfeed-member / AuthenticationService

```
// 블랙리스트 확인
if (BlackListedPredicate.BLACKLISTED.equals(tokenService.isTokenBlacklisted(jti))) {
    log.debug("Token is blacklisted: JTI={}", jti);
    addReLoginRequiredAtResponseHeader(response);
    return;
}
```

주요 정책

Follow 정책, 댓글/답글 정책

- Follow 정책
- 댓글/답글 정책

Follow 정책, 댓글/답글 정책

Follow 정책 - 유저의 최대 follow 가능 멤버 수를 1000명 미만으로 제한

- 한 사람이 1000명 이상을 Follow 하지 못하도록 정의
- 나를 팔로우 하는 팔로워가 100만명이 넘어가는 것은 상관이 없지만, 1000명 이상을 팔로우하는 것은 서버에서는 1000명 이상의 피드를 조회해야 하는 케이스가 발생하기 때문에

답글 허용 레벨 - 답글 3레벨까지만 허용

- 답글 삭제시 중첩 삭제하는 경우를 고려해 3레벨 까지만 허용
- (참고) youtube,instagram 의 경우 커뮤니티 성격의 플랫폼, 답글은 3단계까지만 허용. 대신 멘션 기능을 제공
- (참고) redis, x.com 의 경우 개인의 의견이 중요한 플랫폼, 삭제된 댓글은 '삭제된 댓글입니다'를 표기 후 자식 답글을 표현하고 답글을 모두 펼쳐서 로드, 답글레벨이 무한일 경우에는 가급적 중첩 삭제는 미지원한다는 점

답글 삭제 정책

- 답글 삭제 시 자식 답글은 모두 삭제
- (참고) redis, x.com 의 경우 개인의 의견이 중요한 플랫폼, 삭제된 댓글은 '삭제된 댓글입니다'를 표기 후 자식 답글을 표현

댓글/답글 API 별도 분리

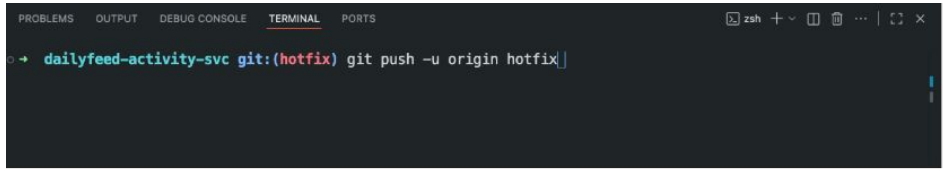
- REST API 의 규칙만 그대로 따를 경우, 댓글 작성 API, 답글 작성 API 를 모두 POST /api/comments 에 두는 것을 생각할 수 있으나 POST /api/comments, POST /api/reply 로 별도 분리
- 댓글/답글 각각의 요청에 대한 기획이나 운영정책은 바뀌기 쉬운 요소이기에, 하나의 API의 변화가 두 기능에 모두 영향을 끼치는 것은 좋지 않은 현상이므로
- 댓글,답글 기능 각각을 POST /api/comments, POST /api/reply 로 별도 분리해서 각 기능을 독립적으로 수정/관리 되도록 구성

CI/CD

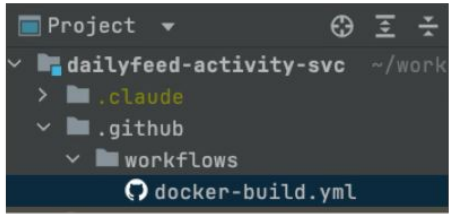
도커 이미지 빌드 자동화 (github workflow)

- 자동 모드
- 수동 모드

자동모드



commit & push



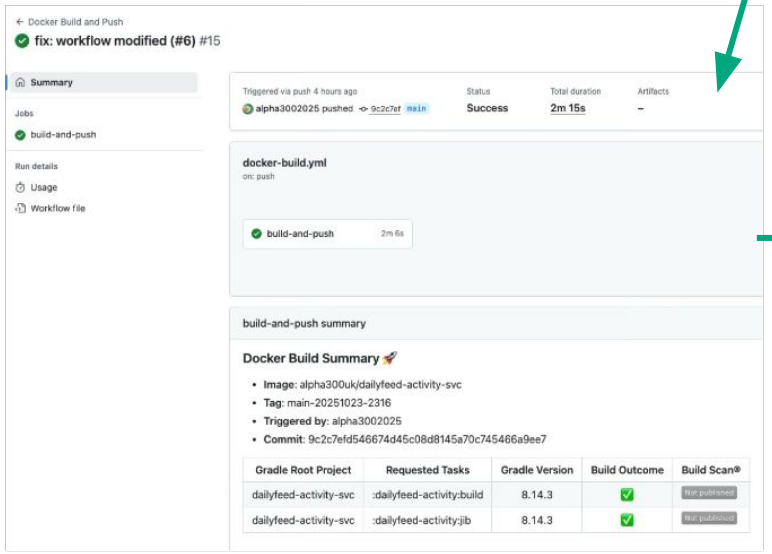
github repository 내의 workflow 발동 (trigger)

참고 :

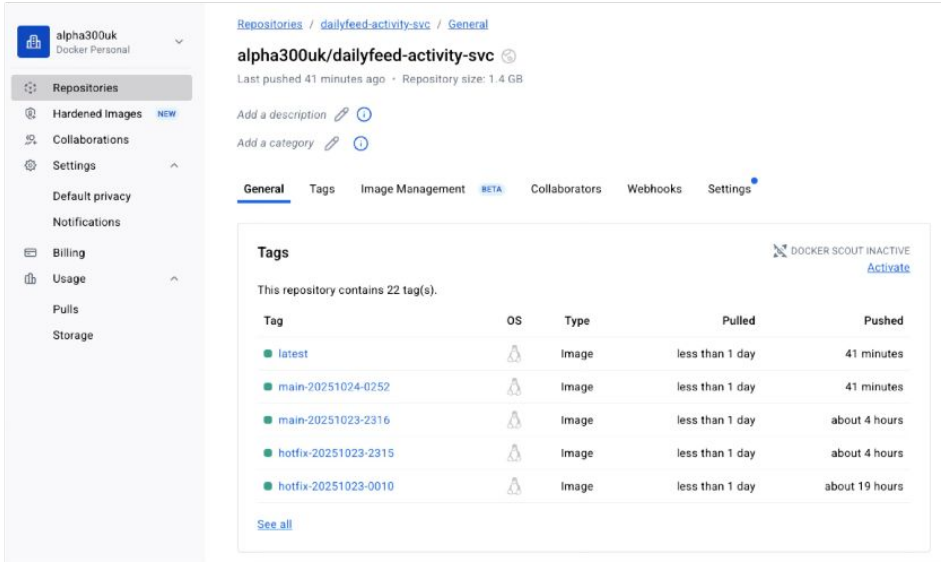
https://github.com/alpha3002025/dailyfeed-activity-svc/blob/main/.github/workflows/docker-build.yml

이미지 태그 형식

{branchName}-{yyyyMMdd}-{build번호}

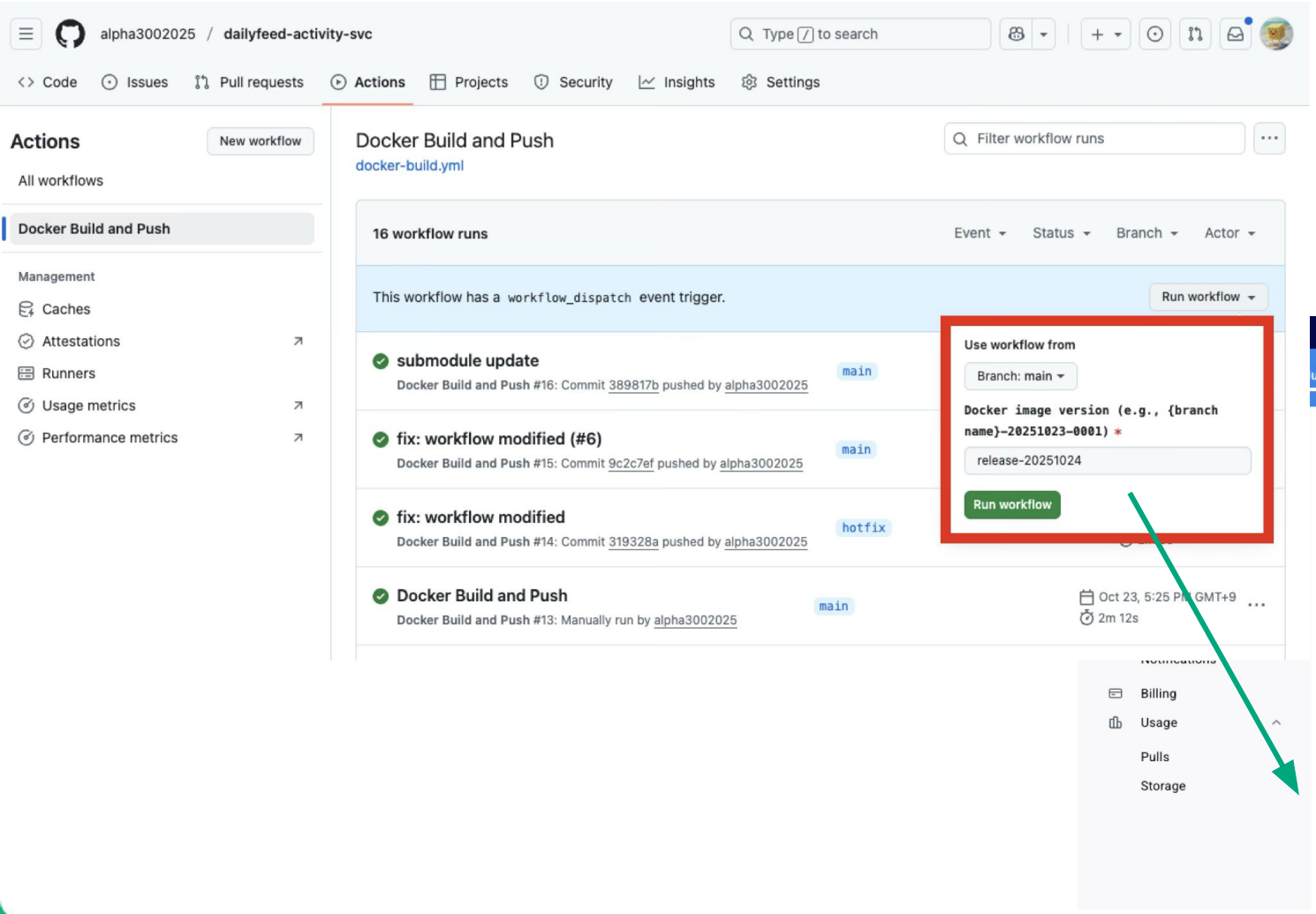


docker 이미지 jib build & push



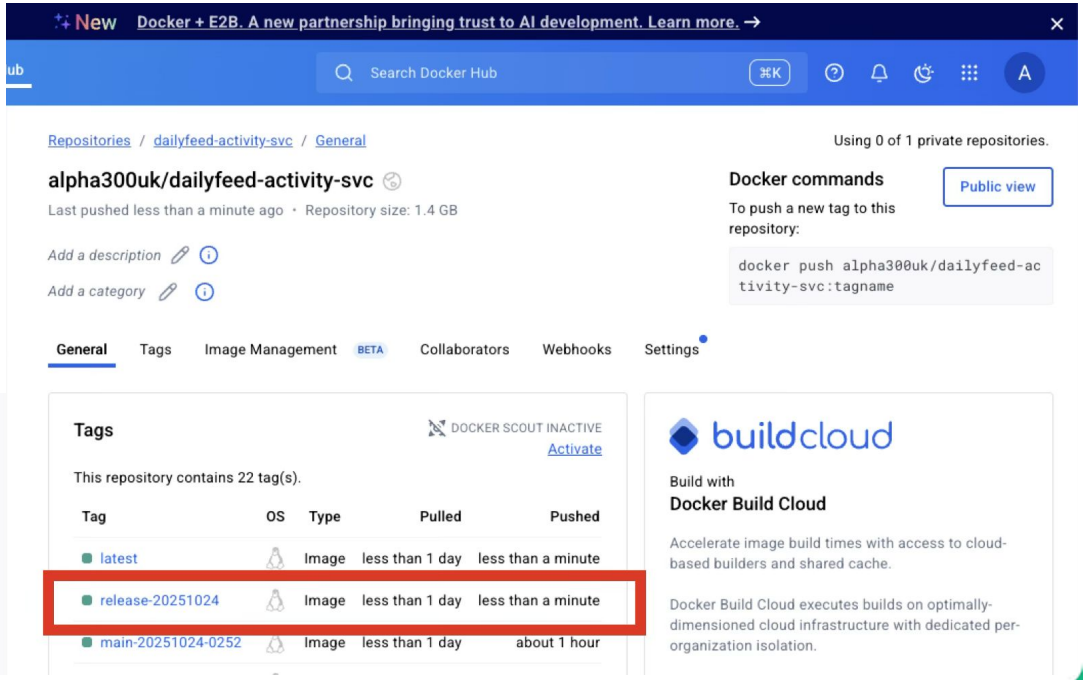
수동모드

github workflow 실행



참고) 현재는 ghcr.io 의 repository 를 사용 중

docker hub (push 된 이미지 확인)



HPA (Horizontal Pod Autoscaler)

HPA 설정

- HPA 설정

HPA 란?

```
→  dailyfeed-installer git:(main) kubectl get po -n dailyfeed
NAME                                  READY   STATUS             RESTARTS      AGE
dailyfeed-activity-69844c4499-5fktm   2/2     Running            0             4m29s
dailyfeed-activity-69844c4499-79lhg   2/2     Running            0             4m14s
dailyfeed-activity-69844c4499-gfhwh   0/2     Init:1/2           0             102s
dailyfeed-content-5f88cc6bb9-5hwx8    0/2     Pending            0             87s
dailyfeed-content-5f88cc6bb9-7km7n    2/2     Running            0             4m15s
dailyfeed-content-5f88cc6bb9-fbhkg    1/2     Running            2 (28s ago)   2m14s
dailyfeed-content-5f88cc6bb9-mwwwh    1/2     CrashLoopBackOff   2 (13s ago)   4m30s
dailyfeed-frontend-5f58b55f68-lgzr6   2/2     Running            0             4m28s
dailyfeed-frontend-5f58b55f68-rck8d   2/2     Running            0             4m28s
dailyfeed-image-79c6459d4b-fgpqd      2/2     Running            0             4m28s
dailyfeed-image-79c6459d4b-qcsc6      2/2     Running            0             4m28s
dailyfeed-image-79c6459d4b-t6846      2/2     Running            0             4m28s
dailyfeed-member-7d59798c58-7qs96     2/2     Running            0             4m15s
dailyfeed-member-7d59798c58-ddqpn     1/2     CrashLoopBackOff   1 (9s ago)    2m59s
dailyfeed-member-7d59798c58-hbxqg     1/2     Running            2 (24s ago)   2m29s
dailyfeed-member-7d59798c58-qkxrk     0/2     Init:1/2           0             102s
dailyfeed-member-7d59798c58-zt2gt     2/2     Running            0             4m30s
dailyfeed-search-5d668444cd-dztlf     2/2     Running            0             4m28s
dailyfeed-search-5d668444cd-k4pct     2/2     Running            0             4m13s
dailyfeed-timeline-84c67c7f89-8x6hk   1/2     CrashLoopBackOff   1 (16s ago)   2m13s
dailyfeed-timeline-84c67c7f89-9rsrb   2/2     Running            0             4m29s
dailyfeed-timeline-84c67c7f89-nn9wx   2/2     Running            0             3m14s
dailyfeed-timeline-84c67c7f89-pj5mn   2/2     Running            0             4m29s
dailyfeed-timeline-84c67c7f89-w7tj8   0/2     Init:1/2           0             16s
```

HPA 가 하는 일

- replication 의 수를 늘리고 줄이는 스케일링(Scaling)을 리소스 메트릭의 특정 수치가 될때 증가/감소될 수 있도록 스케일링을 특정 조건에 따라 자동화하는 역할을 수행
- 트래픽 급증,장애 발생 등의 상황에 단 몇초의 지연도 없이 즉각대응을 통해 스케일링을 하는 것은 불가능에 가까움
- HPA 를 이런 돌발상황에 대한 스케일링 자동화를 걸어두는 작업을 수행

```
→  dailyfeed-installer git:(main) kubectl top nodes
NAME                           CPU(cores)   CPU(%)   MEMORY(bytes)   MEMORY(%)
istio-cluster-control-plane    140m         1%       819Mi           6%
istio-cluster-worker           450m         5%       4677Mi          38%
istio-cluster-worker2          260m         3%       4408Mi          36%
```

metrics-server

- HPA 는 쿠버네티스 클러스터 내의 리소스 들의 metric 의 값이 특정 값의 범위에 있을때 동작하도록 설정하는 역할을 하는데, 이 metric 들의 상태를 수집하는 것이 metric-server 의 역할
- 설치 후 kubectl top nodes, kubectl top nodes -n 네임스페이스 와 같은 명령을 통해 리소스의 상태를 확인 가능

HPA 설정

현재 HPA 설정

서비스	Min/Max Replicas	CPU 임계값	메모리 임계값	Scale Up 특징	비고
activity-svc	2-10	70%	90%	표준 (60초)	표준 설정
content-svc	2-10	70%	90%	표준 (60초)	표준 설정
frontend-svc	2-15	60%	70%	즉시 (0초)	사용자 대면, 가장 공격적
image-svc	2-15	65%	90%	표준 (60초)	CPU/메모리 집약적
member-svc	2-12	70%	90%	표준 (60초)	인증 가용성 중요
search-svc	2-12	65%	90%	표준 (60초)	빠른 응답 필요
timeline-svc	2-10	70%	90%	표준 (60초)	표준 설정

(1) 안정성 우선

모든 서비스에 대해 최소 2개의 replica 부여, scale down 시 안정화윈도우(stabilization window) 를 부여해 급격한 scale down, 플래핑(Flapping) 현상 방지

(2) 서비스 특성 별 세분화

frontend-svc 는 사용자 경험을 위한 빠른 스케일 업(안정화 윈도우 : 0s), image-svc,search-svc 는 CPU 워크로드 임계값을 낮은 값으로 부여

(3) 점진적 스케일링

1분마다 스케일업 검사, 1분마다 임계값 도달시 2개 또는 5개의 Pod 를 스케일업

1분마다 스케일다운 검사, 1분마다 임계값 도달시 2개, 50% 축소

(4) 리소스 requests, limits 세부 정의

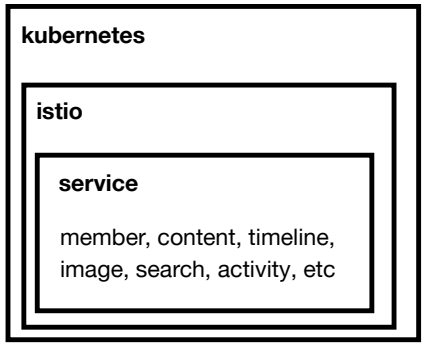
리소스마다 top 명령을 통해 파악한 최소,최대의 범위를 대략적으로 파악 후 적용 (노가다 작업 ^^)

istio 소개

istio 란?

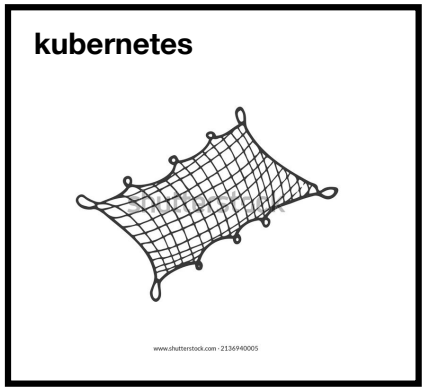
- istio 란?
- Envoy Proxy 란?
- istio 사용의 장점 - (1) HTTPS
- istio 사용의 장점 - (2) Kiali, virtual service, routing
- istio 사용의 장점 - (3) DestinationRule - CircuitBreaker, OutlierDetection
- istio 도입에 대해

istio 란?



Istio

kubernetes 또는 여러 컨테이너 오케스트레이션 등에서 Envoy Proxy 역할을 수행하는 sidecar 를 통해
전체 네트워크의 서비스 메시 (Service Mesh) 레벨에서 라우팅, HTTPS 상호 보안, HttpRetry, Timeout, CircuitBreaker,
HTTP Connection Pool 등의 기능을 제공하는 네트워크 유틸리티 플랫폼 입니다.

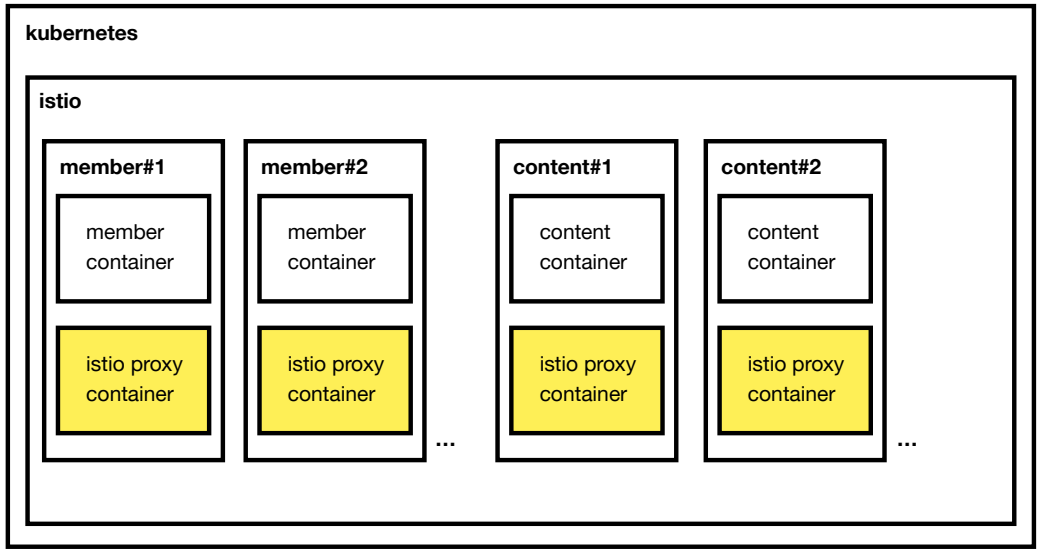


서비스 메시 (Service Mesh) 란?

Mesh 는 network 의 '그물'을 의미하는데, 이 'Service Mesh' 라는 것은 kubernetes 의 Network 레벨을 의미하는 'Mesh'
를 의미합니다. Istio 를 사용하면, 1차적으로 'Mesh'라고 부르는 Network 레벨에서 통합적인 제어 (라우팅, HTTS,
Timeout, HTTP Retry, CircuitBreaker, Connection Pool 등) 를 수행할 수 있습니다.

요청이 너무 많을 경우 그 요청을 모두 처리할 필요가 없을수도 있습니다.
너무 잦은요청은 네트워크 레벨에서 거부하고, 다음 타임윈도우에 재요청하게 하는 것이 나을 수 있습니다.
Service Mesh 는 이런 기능에서부터 다양한 기능들을 제공합니다.
예를 들면 논리적인 라우팅을 통해 여러 버전의 인스턴스를 공존하게 하는 것 역시 가능합니다.

Envoy Proxy 란?



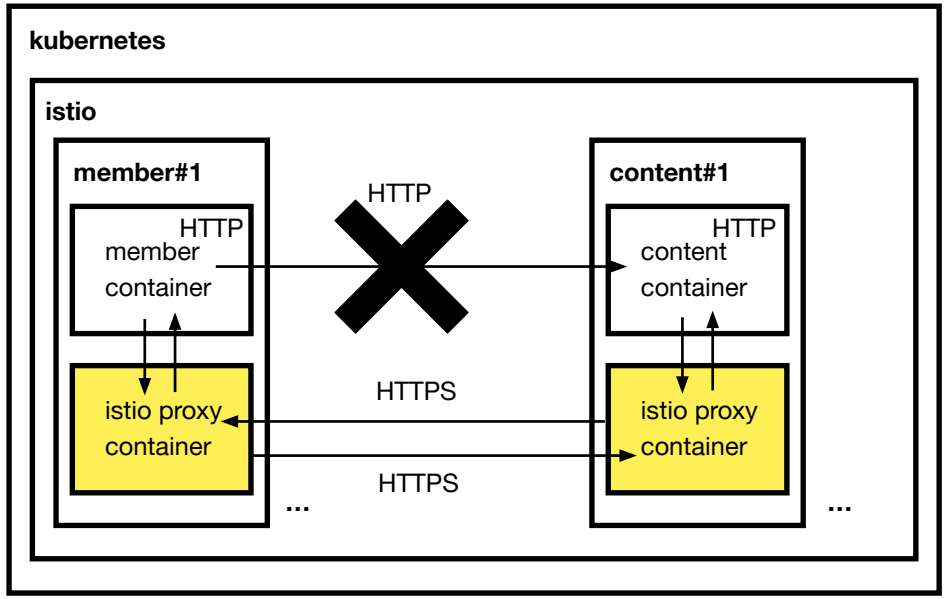
Envoy Proxy, Istio proxy container

좌측 그림의 노란색 상자에는 'istio proxy container' 라는 것이 각각 존재합니다.

istio 는 이렇게 각각의 pod 내에 istio가 관리하는 container 를 주입해서 각각의 pod 에 대한 네트워크 제어를 수행합니다. 이렇게 주입된 'istio proxy container' 들은 각각 자신의 파드를 대표해서 다른 서비스와 통신을 하는 '외교관', 'proxy' 같은 역할을 수행합니다.
예를 들면 HTTPS 가 각각의 pod 에 적용이 안되어 있더라도 istio proxy container 가 주입되어 있고 PeerAuthentication 을 SRICT 로 설정해둔 상태라면, pod 간의 통신시 서비스 컨테이너가 HTTPS 요청을 하지 않더라도 istio proxy container 가 네트워크 레벨에서 HTTPS 암호화를 통해 파드간의 통신을 암호화합니다.

이렇게 각각의 파드에 대해 대사(Envoy) 역할을 하는 Proxy 를 Envoy Proxy 라고 일반적으로 부르며, 일반적으로는 Sidecar 라고 부르기도 하고, Proxy 라고 부르기도 하기도 합니다.

Istio 사용의 장점 - (1) HTTPS



(1) HTTPS (PeerAuthentication)

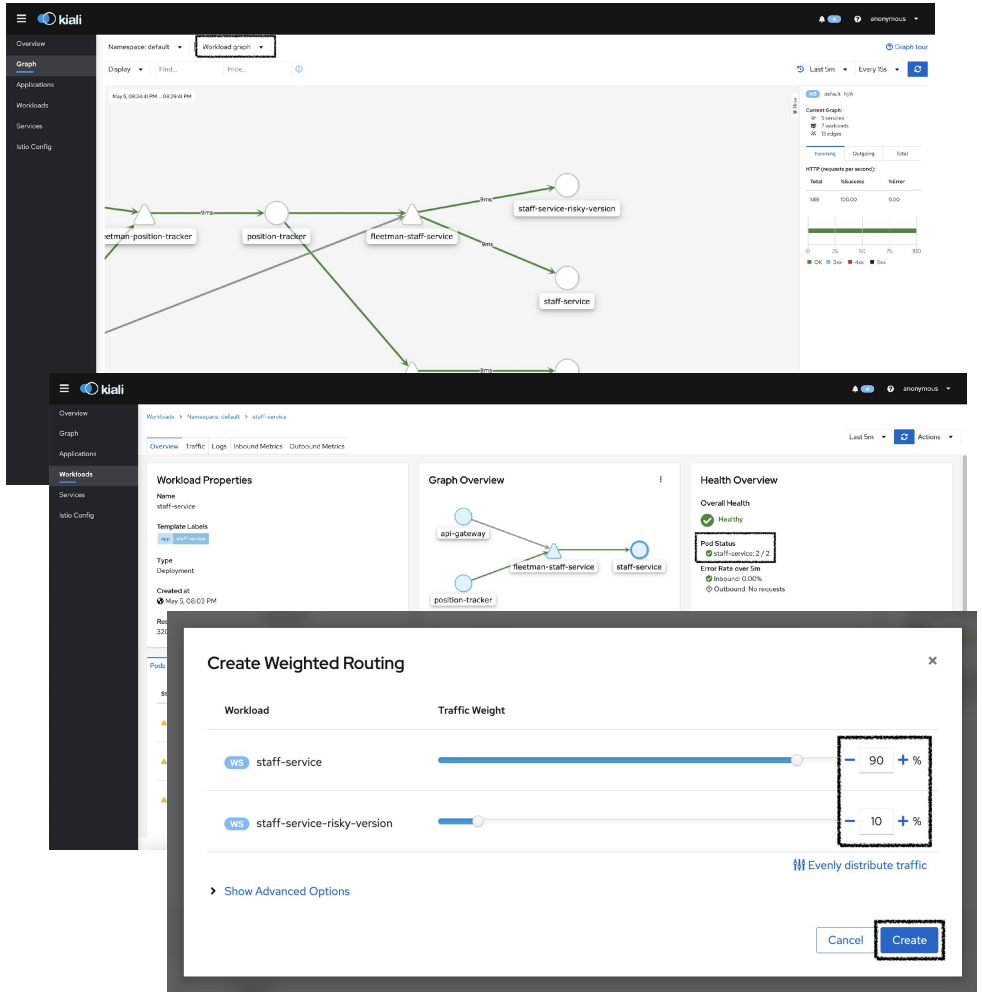
istio 에서는 deployment, pod 내에 배포한 container 가 기본적으로 HTTPS 를 적용하지 않았더라도 istio 의 proxy 는 기본적으로 pod 와 pod 간 통신을 HTTPS 로 암호화해서 통신할 수 있도록 네트워크 레벨에서 암호화를 처리해서 통신을 합니다.

이 설정은 PeerAuthentication 이라는 리소스를 통해 정의되며, 더 세부적인 AuthorizationPolicy 등을 적용하는 것 역시 가능합니다.

kubernetes 를 도입을 하더라도, 각각의 애플리케이션 서비스마다 HTTPS 인증서를 관리하는 것은 쉬운 일이 아닙니다. 만약 Istio 를 사용하게 된다면 이런 작업들을 PeerAuthentication 설정을 통해 최소화 할 수 있게 됩니다.

상황에 따라 pod 와 다른 pod 는 물리적으로 데이터 센터 중 다른 서버 건물 등에 배치될 수도 있습니다. 이 경우 외부와 통신할 때 HTTPS 가 아닌 HTTP 로 통신을 수행할 경우 통신 내용이 평문으로 전달되면서 탈취될 우려가 있는데, Istio 를 사용한다면 혹시라도 모를 1%의 유출 가능성 까지도 모두 암호화를 적용할 수 있다는 점은 장점입니다.

Istio 사용의 장점 - (2) Kiali, virtual service, routing



(2) kiali

istio 에서는 kiali 를 별도의 addon 으로 제공하는데, 각각의 서비스가 어디로 향하는지를 시각화해서 표현한 대시보드를 제공합니다. 이 대시보드 내에서 긴급상황 등에 대해 트래픽의 제어를 할수 있고, 전체 트래픽의 흐름을 확인하는 것 역시 가능합니다.

예를 들어 timeline 애플리케이션의 v2 버전의 디플로이먼트가 슬로우쿼리로 인해 장애가 나고 있다고 해보겠습니다. 이에 대응하기 위해 kiali 대시보드에서는 다음의 대응을 취할수 있습니다.
- v1 의 timeline 디플로이먼트를 배포
- timeline-svc 서비스로의 요청에 대해 v2 버전의 timeline 디플로이먼트로는 요청을 끊기
- timeline-svc 서비스로의 요청에 대해서는 앞으로는 v1 버전의 디플로이먼트로 트래픽이 향하도록 제어

물론 이런 기능은 kiali 에서만 제공되는 기능은 아닙니다. istio 의 yaml 편집을 통해서도 제공할 수 있으며 git 등의 버전관리 등이 적용되도록 일반적인 운영상황에서는 yaml 편집으로 인프라에 대한 내력을 관리합니다.

위와 같은 장애 상황에서만 사용하지는 않습니다. 신규버전 배포시 카나리 배포와 유사한 방식을 istio 내에서 적용할 수 있습니다. 예를 들면 다음과 같은 방식입니다.
- v2 버전 배포
- virtual service 는 5% 의 트래픽은 v2 를 라우팅, 95% 의 트래픽은 v1 을 라우팅
- (1일 후) virtual service 는 10% 의 트래픽은 v2 를 라우팅, 90% 의 트래픽은 v1 을 라우팅
- (7일 후) virtual service 는 50% 의 트래픽은 v2 를 라우팅, 50% 의 트래픽은 v1 을 라우팅
- (30일 후) virtual service 는 100% 의 트래픽은 v2 를 라우팅, 0% 의 트래픽은 v1 을 라우팅

이렇게 istio 를 사용하면 각 네트워크 트래픽에 weight 을 부여해서 virtual service 내에서 각 destination 에 대해 weight 을 부여해서 몇 퍼센트의 네트워크를 v2 로 흘려보내서 장애가 나는지 아닌지를 미리 파악해보면서 천천히 배포해볼수 있게 됩니다.

Istio 사용의 장점 - (3) DestinationRule - CircuitBreaker, OutlierDetection

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: dailyfeed-activity
  namespace: dailyfeed
spec:
  host: dailyfeed-activity
  trafficPolicy:
    # Load balancing: 최소 요청 수를 가진 인스턴스로 라우팅 (기본 ROUND_ROBIN보다 효율적)
    loadBalancer:
      simple: LEAST_REQUEST
      leastRequestLbConfig:
        choiceCount: 2  # 2개 인스턴스 중 최소 요청 선택

    # Connection Pool Settings
    connectionPool:
      tcp:
        maxConnections: 100           # TCP 최대 연결 수
        connectTimeout: 3s            # 연결 타임아웃
        tcpKeepalive:                 # TCP Keepalive 설정
          time: 7200s                 # 2시간
          interval: 75s
          probes: 9
      http:
        http1MaxPendingRequests: 100  # HTTP/1.1 대기 요청 수
        http2MaxRequests: 500         # HTTP/2 최대 요청 수
        maxRequestsPerConnection: 10  # 연결당 최대 요청 수 (연결 재사용 제한)
        maxRetries: 3                 # 최대 재시도 수
        idleTimeout: 30s              # 유휴 연결 타임아웃
        h2UpgradePolicy: UPGRADE      # HTTP/2 업그레이드 허용

    # Outlier Detection (Circuit Breaker)
    outlierDetection:
      consecutive5xxErrors: 5         # 연속 5xx 에러 5회 시 제거
      consecutiveGatewayErrors: 3     # 연속 게이트웨이 에러 3회 시 제거 (더 민감하게)
      interval: 1m                    # 분석 간격
      baseEjectionTime: 5m            # 기본 제거 시간
      maxEjectionPercent: 50          # 최대 제거 비율 (50% 이상 제거 방지)
      minHealthPercent: 30            # 최소 건강한 인스턴스 비율 (30% 보장)
      splitExternalLocalOriginErrors: true  # 외부/내부 오류 구분

  # Subsets (버전별 라우팅용 - 필요시 사용)
  subsets:
  - name: v1
    labels:
      version: v1
```

(3) DestinationRule

Destination Rule 은 VirtualService 가 관리하는 목적지 룰을 의미하는데, 이 Destination Rule 은 Connection Pool 의 갯수부터 어떤 에러가 나타났을 때 잠시 Eject 할지, 얼마 동안 Eject 할지 등을 의미하는 Circuit Breaking, LoadBalaner 기능 까지도 지원합니다.
더 자세한 내용은 좌측의 캡쳐를 참고해주시기 바랍니다.

애플리케이션 레벨에서도 CircuitBreaker 를 제공하는 것 역시 가능합니다.

하지만 혹시라도 모르는 장애 상황에서 CircuitBreaker 가 적용되지 않은 애플리케이션 서비스는 있을수도 있습니다. 항상 모든 애플리케이션에 CircuitBreaker 를 적용할 수 있는 것은 아니기에, 이런 경우에 대해 네트워크 레벨에서 CircuitBreaker 가 작동된다면, 전체 서비스로 장애가 전파되지 않을 수 있다는 안전 대책을 마련할 수 있게 됩니다.

Istio Service Mesh 도입 장점

Istio

Cilium

Linkerd

보안 및 제어 통합

- **mTLS (HTTPS):** 서비스 간 통신 자동 암호화
- **Traffic Control:** 가중치 기반 라우팅 및 카나리 배포
- **Resilience:** Circuit Breaker, Timeout, Retry 설정
- **Observability:** Kiali 대시보드를 통한 시각화

Istio 도입에 대해

Istio 도입에 대해

위에서 살펴봤던 장점들은 istio의 모든 장점을 설명하지는 못합니다.
istio 도입은 필수가 아니지만, 아마도 대부분의 kubernetes 환경에서 제대로된 데브옵스 팀이 운영되고 있다면
아마도 대부분이 istio 를 운영하고 있을 가능성이 큽니다.

쿠버네티스 도입이 배포/장애 관리 등의 이슈로 인해 오히려 운영의 효율성을 떨어뜨릴때도 있는데
이 경우 istio 를 도입한다면 운영상의 어려움을 경감시킬 수 있습니다.

istio 설정 소개

VirtualService, DestinationRule

VirtualService

- DestinationRule 이란?
- DestinationRule 설정 (1)
- DestinationRule 설정 (2)

Virtual Service 설정

VirtualService 는 HttpTimeout, HttpRetry 를 정의

	member	content	timeline	image	search	activity
timeout	10s	10s	10s	30s (이미지 처리는 시간이 오래 걸릴수도 있다는 점을 감안)	15s (검색은 조금 더 긴 타임아웃 필요)	10s
retries/attempts	5	5	5	3 (이미지 업로드는 재시도를 적게하기 위해)	5	5
retries/perTryTimeout	3s	3s	3s	10s (이미지 처리 시간을 고려)	4s	3s
retries/retryOn	gateway-error, connect-failure, refused-stream, 5xx, retriable-4xx, reset	gateway-error, connect-failure, refused-stream, 5xx, retriable-4xx, reset	gateway-error, connect-failure, refused-stream, 5xx, retriable-4xx, reset	gateway-error, connect-failure, refused-stream, reset	gateway-error, connect-failure, refused-stream, 5xx, retriable-4xx, reset	gateway-error, connect-failure, refused-stream, 5xx, retriable-4xx, reset
retries/retryRemoteLocalities	true	true	true	true	true	true

DestinationRule 이란?

Connection Pool & Outlier Detection

Istio의 DestinationRule을 사용하여 트래픽 정책을 관리합니다.

- Connection Pool: TCP/HTTP 연결 수 제한
  (maxConnections: 100)
- HTTP Settings: 최대 대기 요청 수
  (http1MaxPendingRequests: 10)
- Outlier Detection: 5xx 에러 5회 발생 시 1분간 해당
  엔드포인트 차단
- Circuit Breaker: 서비스 가용성 확보를 위한 차단 기법 적용

DestinationRule 설정 (1)

member

connectionPool

maxConnections: 100, connectionTimeout: 3s, tcpKeepalive (time = 7200s, interval = 75s, probes = 9), http1MaxPendingRequests:100, http2MaxRequests:500, maxRequestsPerConnection:10, maxRetries:3, idleTimeout:30s, h2UpgradePolicy: UPGRADE

outlierDetection

consecutive5xxErrors: 5, consecutiveGatewayErrors: 3, interval: 1m, baseEjectionTime: 5m, maxEjectionPercent: 50, minHealthPercent: 30, splitExternalLocalOriginErrors: true

content

connectionPool

maxConnections: 100, connectionTimeout: 3s, tcpKeepalive (time = 7200s, interval = 75s, probes = 9), http1MaxPendingRequests:100, http2MaxRequests:500, maxRequestsPerConnection:10, maxRetries:3, idleTimeout:30s, h2UpgradePolicy: UPGRADE

outlierDetection

consecutive5xxErrors: 5, consecutiveGatewayErrors: 3, interval: 1m, baseEjectionTime: 5m, maxEjectionPercent: 50, minHealthPercent: 30, splitExternalLocalOriginErrors: true

timeline

connectionPool

maxConnections: 100, connectionTimeout: 3s, tcpKeepalive (time = 7200s, interval = 75s, probes = 9), http1MaxPendingRequests:100, http2MaxRequests:500, maxRequestsPerConnection:10, maxRetries:3, idleTimeout:30s, h2UpgradePolicy: UPGRADE

outlierDetection

consecutive5xxErrors: 5, consecutiveGatewayErrors: 3, interval: 1m, baseEjectionTime: 5m, maxEjectionPercent: 50, minHealthPercent: 30, splitExternalLocalOriginErrors: true

search

connectionPool

maxConnections: 120, connectionTimeout: 3s, tcpKeepalive (time = 7200s, interval = 75s, probes = 9), http1MaxPendingRequests:120, http2MaxRequests:550, maxRequestsPerConnection:10, maxRetries:3, idleTimeout:45s, h2UpgradePolicy: UPGRADE

outlierDetection

consecutive5xxErrors: 5, consecutiveGatewayErrors: 3, interval: 1m, baseEjectionTime: 5m, maxEjectionPercent: 50, minHealthPercent: 30, splitExternalLocalOriginErrors: true

DestinationRule 설정 (2)

image

connectionPool

maxConnections: 150, connectionTimeout: 3s, tcpKeepalive (time = 7200s, interval = 75s, probes = 9), http1MaxPendingRequests:150, http2MaxRequests:600, maxRequestsPerConnection:5, maxRetries:2, idleTimeout:60s, h2UpgradePolicy: UPGRADE

outlierDetection

consecutive5xxErrors: 5, consecutiveGatewayErrors: 3, interval: 1m, baseEjectionTime: 5m, maxEjectionPercent: 50, minHealthPercent: 30, splitExternalLocalOriginErrors: true

activity

connectionPool

maxConnections: 100, connectionTimeout: 3s, tcpKeepalive (time = 7200s, interval = 75s, probes = 9), http1MaxPendingRequests:10, http2MaxRequests:500, maxRequestsPerConnection:10, maxRetries:3, idleTimeout:30s, h2UpgradePolicy: UPGRADE

outlierDetection

consecutive5xxErrors: 5, consecutiveGatewayErrors: 3, interval: 1m, baseEjectionTime: 5m, maxEjectionPercent: 50, minHealthPercent: 30, splitExternalLocalOriginErrors: true

Thank You

Dailyfeed Backend Architecture Overview

alpha300uk@gmail.com