11장. 뉴스 피드 시스템 설계

6## 1단계 | 문제 이해 및 설계 범위 확정

- 사용자는 뉴스 피드 페이지에 새로운 스토리를 올릴 수고, 친구들이 올린 스토리를 볼 수 있어야 한다.
- 시간 흐름 역순으로 스토리가 표시되어야 한다.
- 한 명의 사용자는 최대 5000명의 팔로워를 가질 수 있다.
- 하루에 천만 명이 방문할 수 있다.
- 피드에는 이미지나 비디오 등의 미디어 파일이 포함될 수 있다.

2단계 | 개략적 설계안 제시 및 동의 구하기

지금부터 살펴 볼 설계안은 피드 발행, 뉴스 피드 생성의 두 가지 부분으로 나뉘어 있다.

- 피드 발행: 사용자가 스토리를 포스팅하면 해당 데이터를 캐시와 데이터베이스에 기록한다. 새 포스팅은 친구의 뉴스 피드에도 전송된다.
- 뉴스 피드 생성: 지면 관계상 뉴스 피드는 모든 친구의 포스팅을 시간 흐름 역순으로 모아서 만든다고 가정한다.

뉴스 피드 API

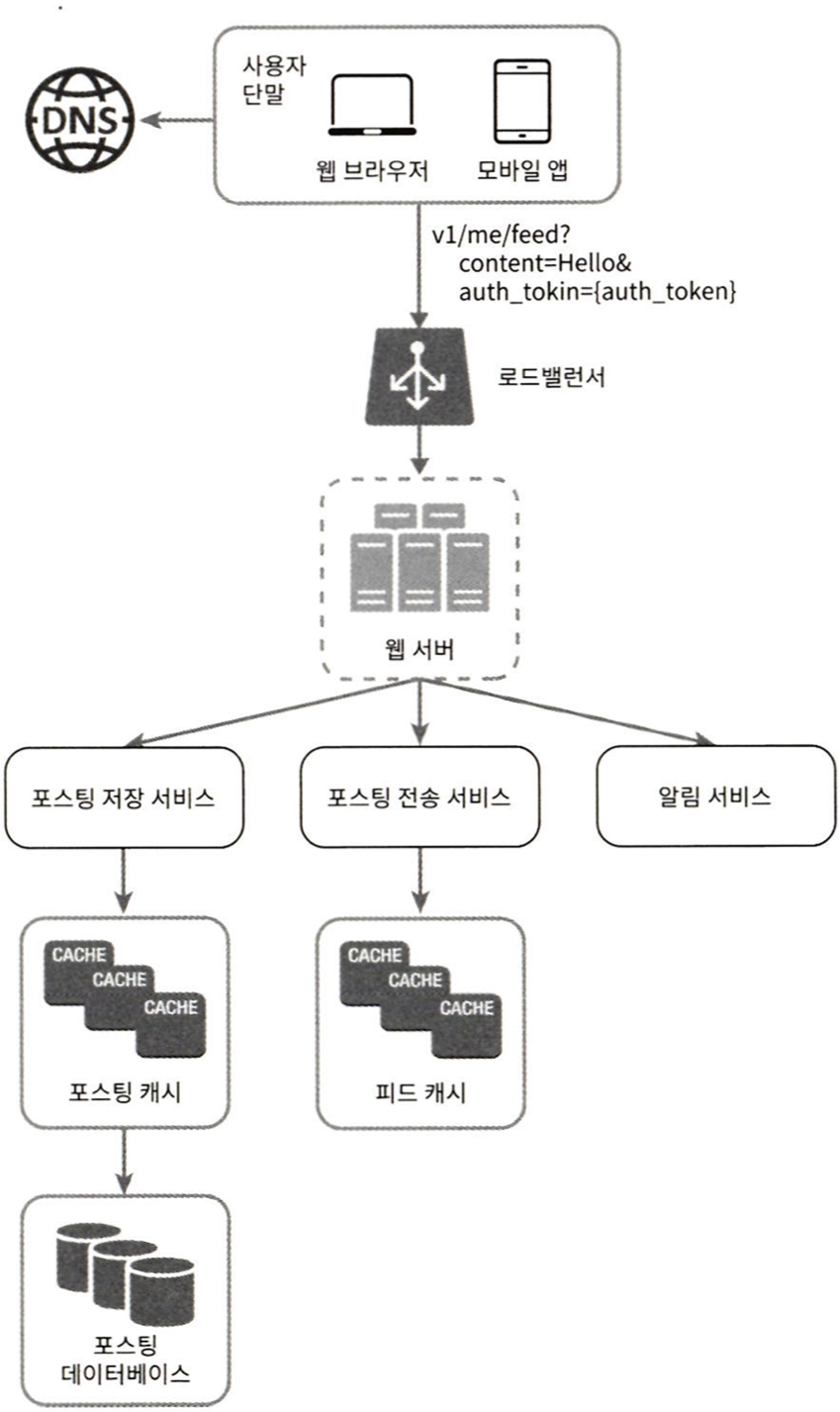
클라이언트가 서버와 통신하기 위해 사용하는 수단

- HTTP 프로토콜 기반
- 상태 정보를 업데이트하거나, 뉴스 피드를 가져오거나, 친구를 추가하는 등 다양한 작업을 수행하는 데 사용

피드 발행 API

새 스토리를 포스팅하기 위한 API
`POST /v1/me/feed`
인자

- body: 포스팅 내용에 해당
- Authorization 헤더: API 호출을 인증하기 위해 사용



- 사용자: 모바일 앱이나 브라우저에서 새 포스팅을 올리는 주체
- 로드밸런서: 트래픽을 웹 서버들로 분산
- 웹 서버: HTTP 요청을 내부 서비스로 중계하는 역할
- 포스팅 저장 서비스: 새 포스팅을 데이터베이스와 캐시에 저장
- 포스팅 전송 서비스: 새 포스팅을 친구의 뉴스 피드에 푸쉬(데이터는 캐시에 보관하여 빠르게 읽어갈 수 있도록 함)
- 알림 서비스: 친구들에게 새 포스팅이 올라왔음을 알리거나, 푸시 알림을 보내는 역할을 담당

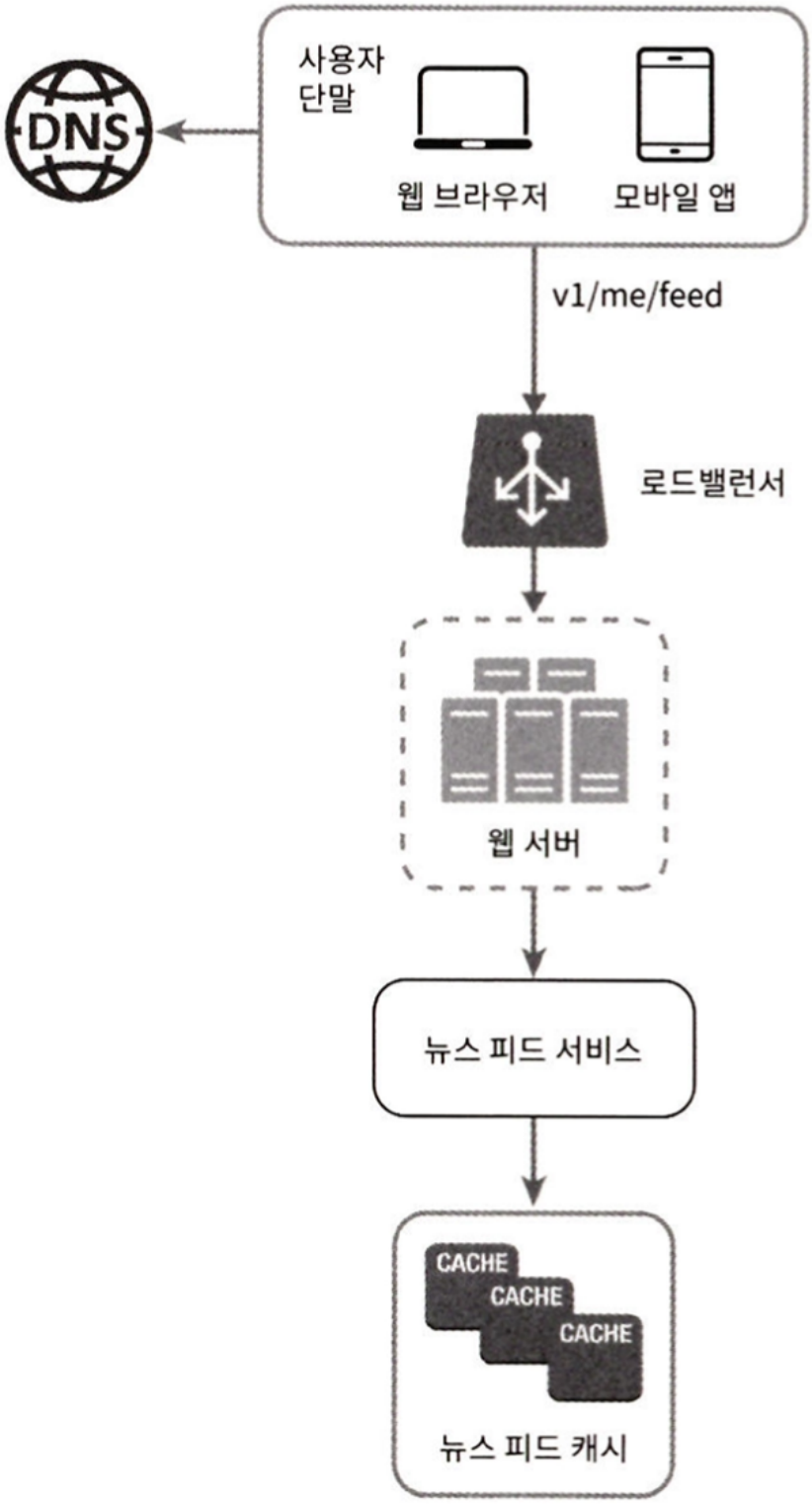
피드 읽기 API

뉴스 피드를 가져오는 API

`GET /v1/me/feed`

인자

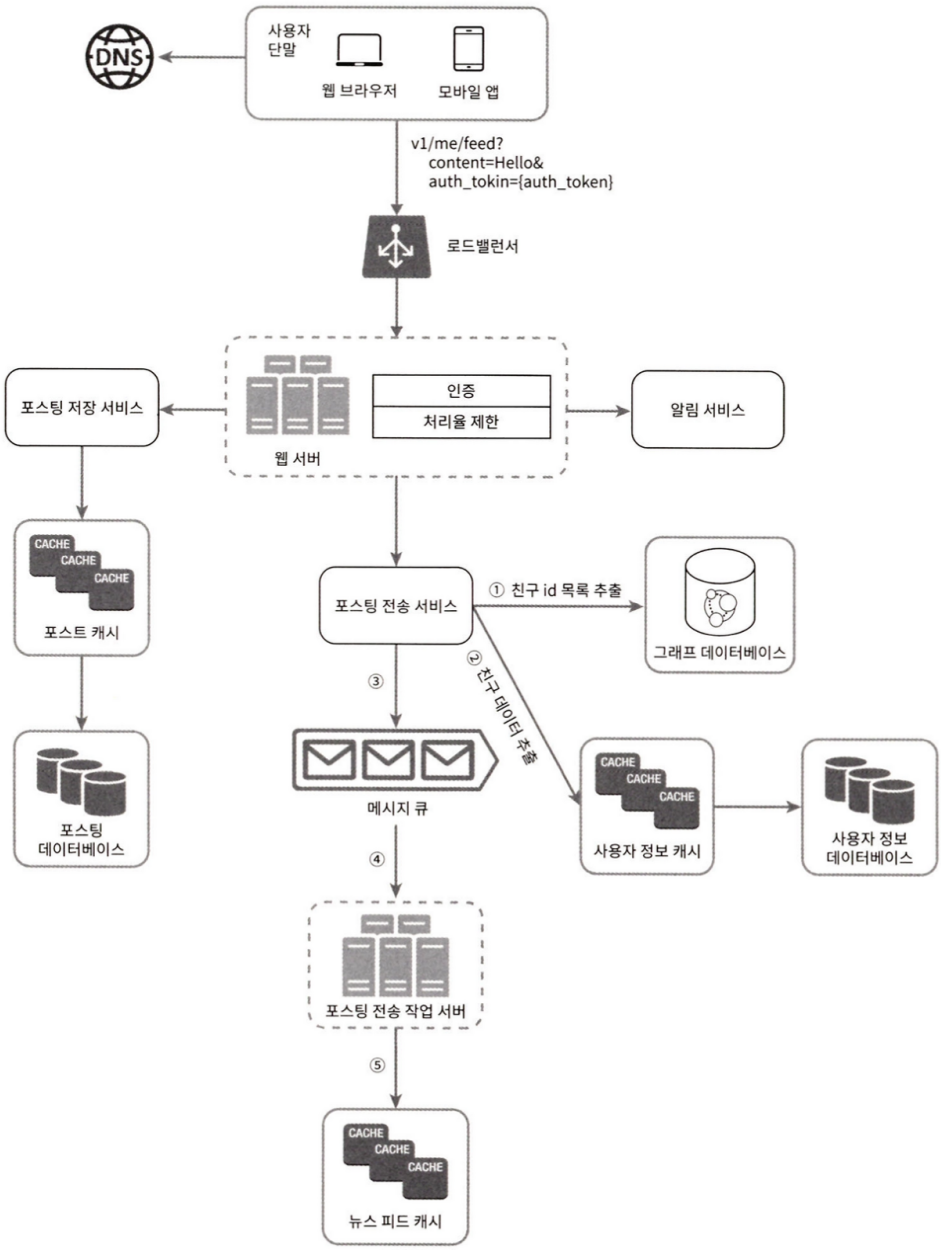
- Authorization 헤더: API 호출을 인증하기 위해 사용



- 사용자: 뉴스 피드를 읽는 주체
- 로드밸런서: 트래픽을 웹 서버들로 분산
- 웹 서버: 트래픽을 뉴스 피드 서비스로 전송
- 뉴스 피드 서비스: 캐시에서 뉴스 피드를 가져오는 서비스
- 뉴스 피드 캐시: 뉴스 피드를 랜더링할 때 필요한 파드 ID를 보관

3단계 | 상세 설계

피드 발생 흐름 상세 설계



웹 서버

- 클라이언트와 통신
- 인증
- 처리율 제한

포스팅 전송(팬아웃) 서비스

포스팅 전송, 즉 팬아웃은 어떤 사용자의 새 포스팅을 그 사용자와 친구 관계에 있는 모든 사용자에게 전달하는 과정

- 팬아웃 모델

- 쓰기 시점에 팬아웃하는 모델(fanout-on-write) == 푸쉬 모델
- 읽기 시점에 팬아웃하는 모델(fanout-on-read) == 풀 모델

쓰기 시점에 팬아웃하는 모델

새로운 포스팅을 기록하는 시점에 뉴스 피드를 갱신하게 된다. 다시 말해, 포스팅이 완료되면 바로 해당 사용자의 캐시에 해당 포스팅을 기록하는 것이다.

- 장점
  - 뉴스 피드가 실시간으로 갱신되며 친구 목록에 있는 사용자에게 즉시 전송된다.
  - 포스팅이 쓰이는 시점에 전송되므로 뉴스 피드를 읽는 데 드는 시간이 짧아진다.
- 단점
  - 친구가 많은 사용자의 경우 뉴스 피드 갱신에 많은 시간이 소요될 수 있다. **핫키**라고 부르는 문제이다.
  - 사용자의 친구 목록에 있는 사용자의 피드를 모두 갱신하므로 서비스를 자주 이용하지 않는 사용자의 피드까지 갱신된다. 따라서 컴퓨팅 자원이 낭비된다.

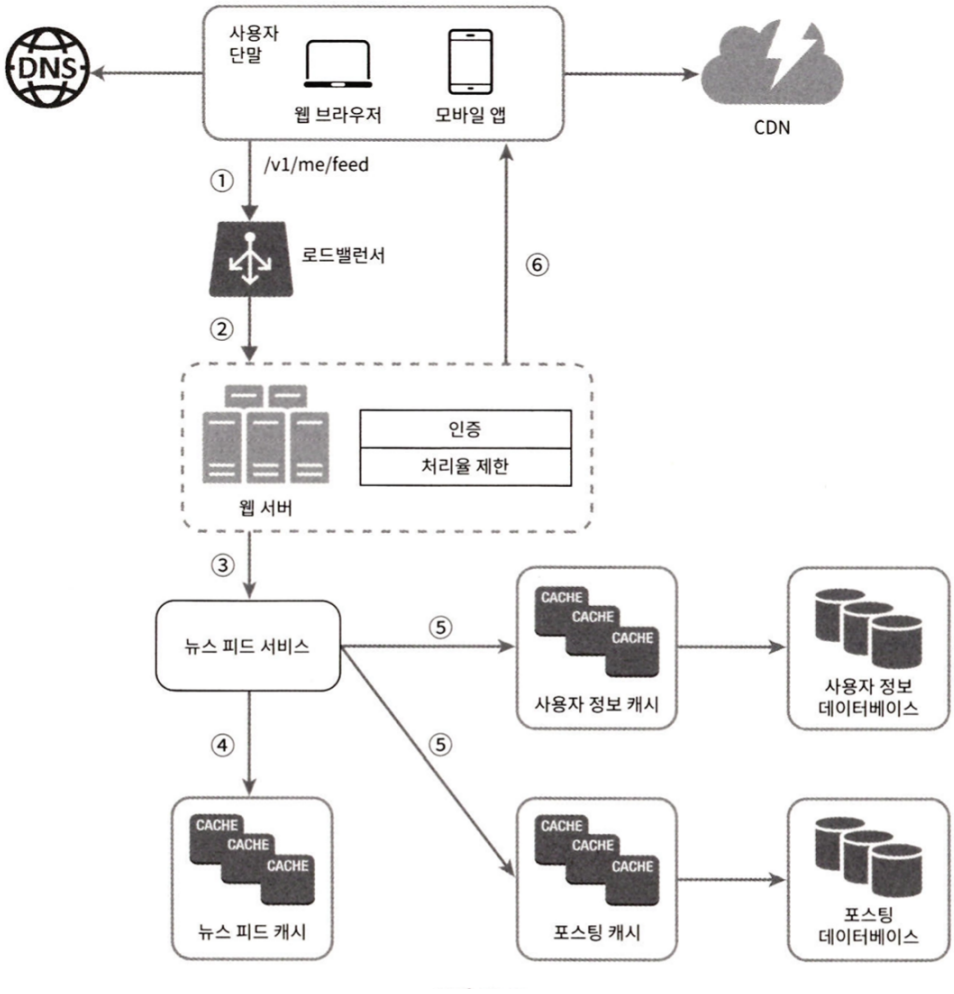
읽기 시점에 팬아웃하는 모델

피드를 읽어야 하는 시점에 뉴스 피드를 갱신한다. 따라서 요청 기반 모델이다.

- 장점
  - 비활성화된 사용자, 또는 서비스에 거의 로그인하지 않는 사용자의 경우네는 이 모델이 유리하다. 로그인하기까지는 어떤 컴퓨팅 자원도 소모하지 않아서다.
  - 데이터를 친구 목록의 사용자 모두에게 푸시하는 작업이 필요 없으므로 핫키 문제도 발생하지 않는다.
- 단점
  - 뉴스 피드를 읽는데 많은 시간이 소요될 수 있다.

본 설계안의 경우에는 이 두 가지 방법을 결합하여 장점은 취하고 단점은 버리는 전략을 취함

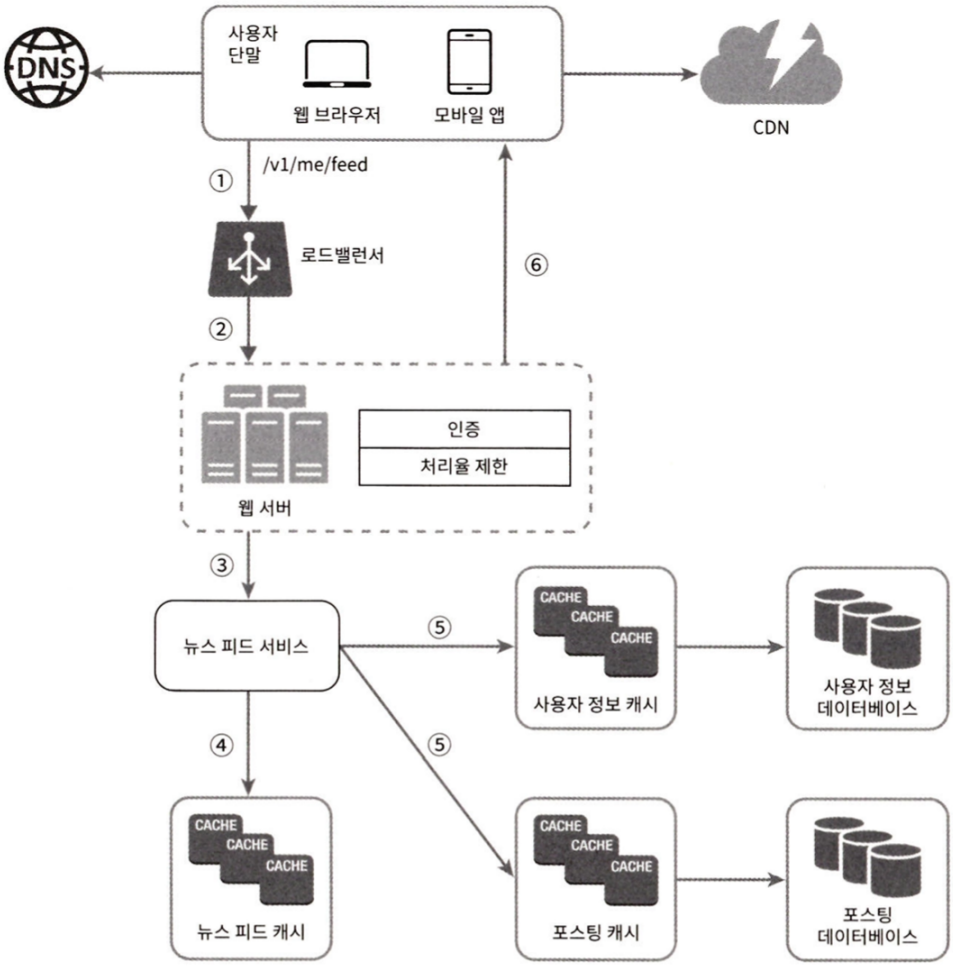
- 뉴스 피드를 빠르게 가져올 수 있도록 하는 것은 아주 중요 → 대부분의 사용자에 대해 **푸시 모델** 사용
- 친구나 팔로워가 아주 많은 사용자의 경우 → 시스템 부하 방지를 위해 **풀 모델**을 사용
- 안정 해시를 통해 요청과 데이터를 보다 고르게 분산하여 핫키 문제를 줄여볼 것



동작 과정

1. 그래프 데이터베이스에서 친구 ID 목록을 가져온다.
2. 사용자 정보 캐시에서 친구들의 정보를 가져온다. 그런 후에 사용자 설정에 따라 친구 가운데 일부를 걸러낸다.(차단한 친구)
3. 친구 목록과 새 스토리의 포스팅 ID를 메시지 큐에 넣는다.
4. 팬아웃 작업 서버가 메시지 큐에서 데이터를 꺼내어 뉴스 피드 데이터를 뉴스 피드 캐시에 넣는다. 뉴스 피드 캐시는 <포스팅 ID, 사용자 ID>의 순서쌍을 보관하는 매핑 테이블이라고 볼 수 있다.
    - 어떤 사용자가 뉴스 피드에 올라온 수천 개의 스토리를 전부 훑어보는 일이 벌어질 확률은 지극히 낮다. 대부분의 사용자가 보려 하는 것은 최신 스토리다. 따라서 캐시 미스가 일어날 확률은 낮다.

피드 읽기 흐름 상세 설계



동작 과정

1. 사용자가 뉴스 피드를 읽으려는 요청을 보낸다.
2. 로드밸런서가 요청을 웹 서버 가운데 하나로 보낸다.
3. 웹 서버는 피드를 가져오기 위해 뉴스 피드 서비스를 호출한다.
4. 뉴스 피드 서비스는 뉴스 피드 캐시에서 포스팅 ID 목록을 가져온다.
5. 뉴스 피드에 표시할 사용자 이름, 사용자 사진, 포스팅 콘텐츠, 이미지 등을 사용자 캐시와 포스팅 캐시에서 가져와 완전한 뉴스 피드를 만든다.
6. 생성된 뉴스 피드를 JSON 형태로 클라이언트에게 보낸다. 클라이언트는 해당 피드를 랜더링한다.

캐시 구조

캐시는 뉴스 피드 시스템의 핵심 컴포넌트다. 본 설계안의 경우에는 아래의 그림과 같이 캐시를 다섯 계층으로 나눈다.

- 뉴스 피드: 뉴스피드의 ID를 보관
- 콘텐츠: 포스팅 데이터를 보관하고 인기 콘텐츠는 따로 보관
- 소셜 그래프: 사용자 간 관계 정보를 보관 (팔로워, 팔로잉)
- 행동: '좋아요'나 댓글 같은 사용자 행위에 관한 정보를 보관
- 횟수: '좋아요' 횟수, 응답 수, 팔로워 수, 팔로잉 수 등의 정보를 보관

4단계 | 마무리

다루면 좋을 만한 주제

- 수직적 규모 확장 vs 수평적 규모 확장
- SQL vs NoSQL
- 주-부 데이터베이스 다중화
- 복제본에 대한 읽기 연산
- 일관성 모델
- 데이터베이스 샤딩

논의해 보면 좋을 만한 주제

- 웹 계층을 무상태로 운영하기
- 가능한 한 많은 데이터를 캐시할 방법
- 여러 데이터 센터를 지원할 방법
- 메시지 큐를 사용하여 컴포넌트 사이의 결합도 낮추기
- 핵심 메트릭에 대한 모니터링. 예를 들어 트래픽이 몰리는 시간대의 QPS, 사용자가 뉴스 피드를 새로고침할 때의 지연시간 등이 이에 해당된다.