11장 뉴스 피드 시스템 설계

뉴스 피드 (news feed) 란?

- “뉴스 피드는 여러분의 홈 페이지 중앙에 지속적으로 업데이트되는 스토리들로, 사용자 상태 정보 업데이트, 사진, 비디오, 링크, 앱활동, 그리고 여러분이 페이스북에서 팔로우하는 사람들, 페이지, 또는 그룹으로부터 나오는 ‘좋아요’ 등을 포함한다.” (페이스북 도움말)

1단계 문제 이해 및 설계 범위 확정

Q. 지원 방식?

- 모바일 앱, 웹 둘 다 지원.

Q. 중요한 기능?

- 사용자는 뉴스 피드 페이지에 새로운 스토리 올릴 수 있어야 하고, 친구들이 올리는 스토리를 볼 수도 있어야 한다.

Q. 뉴스 피드에는 어떤 순서로 스토리 표시?

- 시간 흐름 역순 (reverse chronological order).

Q. 한 명의 사용자는 최대 몇 명의 친구를 가질 수 있는가?

- 5000 명.

Q. 트래픽 규모는 어느 정도?

- 매일 천 만명 방문 가정.

Q. 피드에 이미지나 비디오 스토리도 올릴 수 있는가?

- 스토리에 이미지나 비디오 등 미디어 팔이 포함될 수 있다.

2단계 계략적 설계안 제시 및 동의 구하기

설계안 분류

- 피드 발행 : 사용자가 스토리를 포스팅하면 해당 데이터를 캐시와 데이트베이스에 기록.
- 뉴스 피드 생성 : 지면 관계상 뉴스 피드는 모든 친구의 포스팅을 시간 흐름 역순으로 모아서 만든다고 가정.

뉴스 피드 API

피드 발행 API

새 스토리를 포스팅하기 위한 API. HTTP POST 형태로 요청을 보냄.

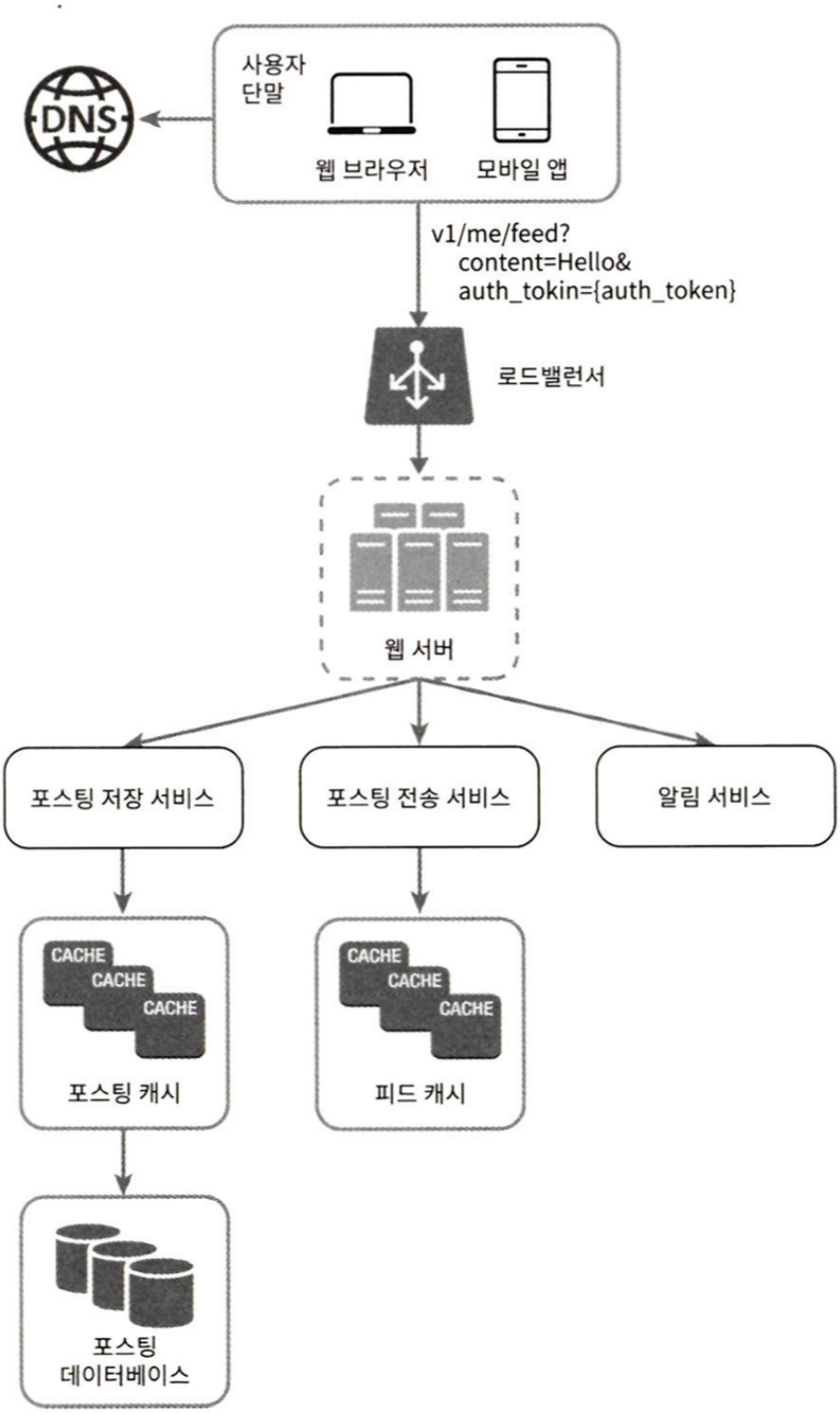
- 형태 : `POST/v1/me/feed`
- 인자
  - 바디 : 포스팅 내용.
  - Authorization 헤더

피드 읽기 API

뉴스 피드를 가져오는 API.

- 형태 : `GET/v1/me/feed`
- 인자
  - Authorization 헤더

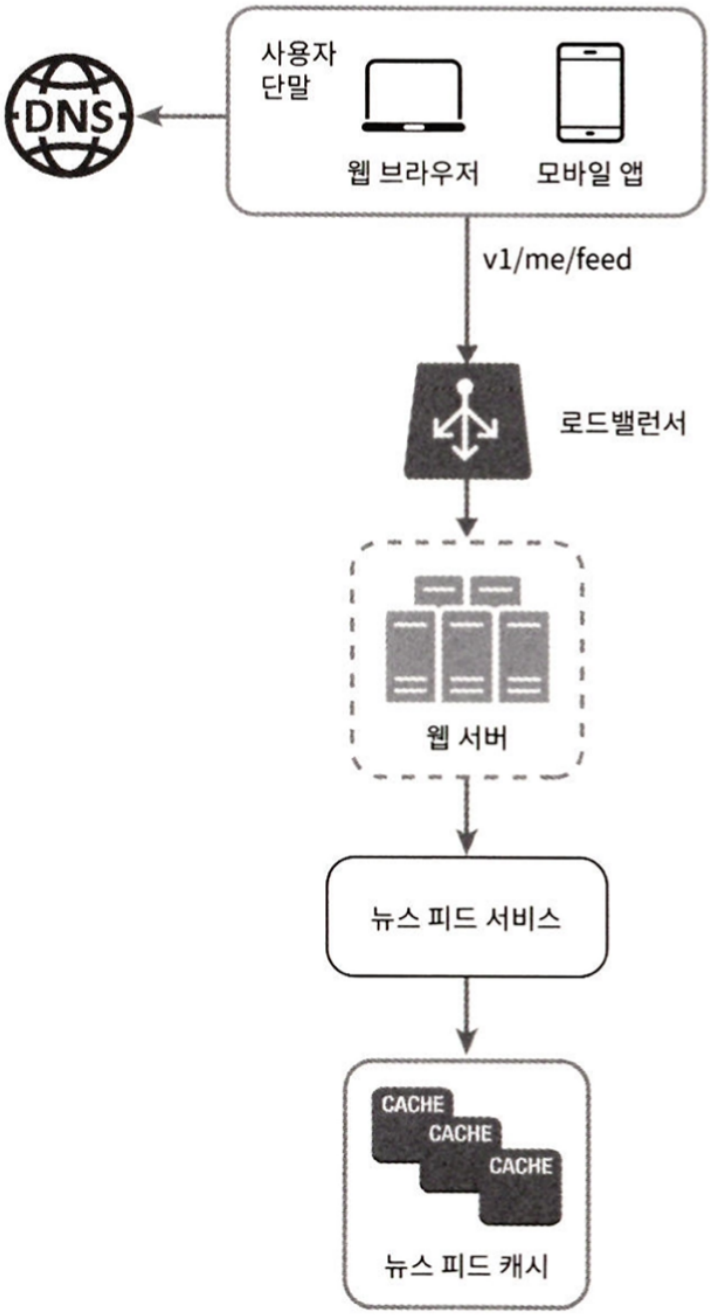
피드 발행



1. 사용자가 모바일 앱이나 브라우저에 새 포스팅을 올림.
2. 로드밸런서가 트래픽을 웹 서버들로 분산
3. 웹 서버가 HTTP 요청을 내부 서비스로 중계
4. 포스팅 저장 서비스롤 통해 새 포스팅을 DB와 캐시에 저장

5. 포스팅 전송 서비스를 통해 새 포스팅을 친구의 뉴스 피드에 푸시. 뉴스 피드 데이터는 캐시에 보관하여 빠르게 읽어갈 수 있도록.
6. 알림 서비스를 통해 친구들에게 새 포스팅일 올라왔음을 알리거나 푸시 알림을 보냄.

뉴스 피드 생성



1. 사용자가 읽기 API 요청을 보냄.
2. 로드 밸런서가 웹 서버들로 트래픽 분산

3. 웹 서버가 트래픽을 뉴스 피드 서비스로 보냄
4. 뉴스 피드 서비스를 통해 캐시에서 뉴스 피드 가져옴.
5. 뉴스 피드 캐시에 뉴스 피드를 랜더링할 때 필요한 피드 ID 를 보관.

3단계 상세 설계

피드 발생 흐름 상세 설계

웹 서버

- 클라이언트와 통신 뿐만 아니라 인증이나 처리율 제한 등의 기능도 수행.
- 올라른 Auth 토큰을 헤더에 넣고 API를 호출하는 사용자만 포스팅할 수 있어야.
- 스팸과 유해한 콘텐츠 노출을 방지하기 위해 특정 기간 동안 한 사용자가 올릴 수 있는 포스팅 수에 제한을 두어야.

포스팅 전송(팬아웃) 서비스

- 팬 아웃 (fanout)은 어떤 사용자의 새 포스팅을 그 사용자와 친구 관계에 있는 모든 사용자에게 전달하는 과정.
- **쓰기 시점 팬아웃 (fan-out-on-write) 모델** : 새로운 포스팅을 기록하는 시점에 뉴스 피드를 갱신. 포스팅을 완료되면 해당 사용자의 캐시에 해당 포스팅 기록.
- 장점
    - 뉴스 필드가 실시간으로 갱신되면 친구 목록에 있는 사용자에게 즉시 전송됨.
    - 새 포스팅이 기록되는 순간에 뉴스 필드가 이미 갱신되므로 뉴스 피드를 읽는데 들이는 시간이 짧아짐.
- 단점
    - 핫키 (hotkey) 문제 : 친구가 많으면 친구 목록을 가져오고 목록의 모든 사용자의 뉴스 피드를 갱신하는 데 많은 시간이 소요될 수 있다.
    - 서비스를 자주 이용하지 않는 사용자의 피드까지 갱신해야 하므로 컴퓨팅 자원 낭비.
- 읽기 시점 팬아웃 (fan-out-on-read) 모델 : 피드를 읽어야 하는 시점에 뉴스 피드를 갱신. 요청 기반 (on-demand) 모델.

- 장점
  - 비활성화된 사용자, 또는 서비스에 거의 로그인하지 않는 사용자의 경우에는 이 모델이 유리. 로그인하기끼지는 어떤 컴퓨팅 자원도 소모하지 않으므로.
  - 데이터를 친구 각각에 푸시하는 작업이 필요 없으므로 핫키 문제도 없음.
- 단점
  - 뉴스 피드를 읽는 데 많은 시간이 소요될 수 있다.
- 모델에 적용
  - 뉴스 피드를 빠르게 가져오는 것은 중요하므로 대부분의 사용자에 대해서 푸시 모델 사용.
  - 친구나 팔로워가 매우 많은 사용자는 팔로워도 하여금 해당 사용자의 포스팅을 필요할 때 가져가도록 하는 풀 모델을 사용하여 시스템 과부하 방지.
  - 안정 해시를 통해 요청과 데이터를 보다 고르게 분산하여 핫키 문제 보다 줄임.
- 작동 프로세스
  1. 그래프 DB에서 친구 ID 목록을 가져온다. 그래프 DB는 친구 관계나 친구 추천을 관리하기 적합하다.
  2. 사용자 정보 캐시에서 친구들의 정보를 가져온다. 그런 후 사용자 설정에 따라 친구 가운데 일부를 걸러낸다.
  3. 친구 목록과 새 스토리의 포스팅 ID를 메시지 큐에 넣는다.
  4. 팬아웃 작업 서버가 메시지 큐에서 데이터를 꺼내 뉴스 피드 데이터를 뉴스 피드 캐시에 넣는다. <포스팅 id, 사용자 id> 의 순서쌍이 캐시에 보관된다. 하지만 메모리 요구량을 고려해 사용자 정보를 모두 넣지는 않는다. 그래서 id만 따로 보관한다.

     또한 메모리 크기를 적정으로 유지하기 위해 캐시 크기에 제한을 두며, 해당 값을 조정할 수 있도록 한다.

피드 읽기 흐름 상세 설계

1. 사용자가 뉴스 피드 읽기 요청을 보낸다.
2. 로드밸런서가 요청을 서버 가운데 하나로 보낸다.
3. 서버는 피드를 가져오기 위해 뉴스 피드 서비스를 호출한다.
4. 뉴스 피드 서비스는 뉴스 피드 캐시에서 포스팀 id 목록을 가져온다.
5. 뉴스 필드에 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.

캐시 구조

- 뉴스 피드 : 뉴스 피드의 id를 보관.
- 콘텐츠 : 포스팅 데이터를 보관. 인기 콘텐츠는 따로 보관.
- 소셜 그래프 : 사용자 간 관계 정보 보관.
- 행동 (action) : 포스팅에 대한 사용자의 행위에 관한 정보 보관. 좋아요, 답글 등
- 횟수 (counter) : 좋아요 횟수, 응답 수, 팔로워 수, 팔로잉 수 등의 정보 보관.

12장 채팅 시스템 설계

1단계 문제 이해 및 설계 범위 확장

Q. 어떤 채팅 지원?

- 1:1 채팅, 그룹 채팅 모두 지원.

Q. 앱 지원 형태?

- 모바일, 웹 앱 둘 다 지원.

Q. 처리해야 하는 트래픽 규모?

- 일별 능동 사용자 (DAU) 기준으로 5천만 명 처리할 수 있어야.

Q. 그룹 채팅은 인원 제한이 있는가?

- 최대 100명까지 참여 가능.

Q. 중요 기능?

- 1:1 채팅, 그룹 채팅, 사용자 접속상태 표시 지원. 텍스트 메시지만 주고 받을 수.

Q. 메시지 길이 제한?

- 100,000자 이하.

Q. 종단 간 암호화 지원?

- 현재는 필요 없지만 시간이 되면 논의 가능.

Q. 채팅 이력 보관 기간?

- 영원히.

2단계 개략적 설계안 제시 및 동의 구하기

클라이언트와 채팅 서비스 간의 관계

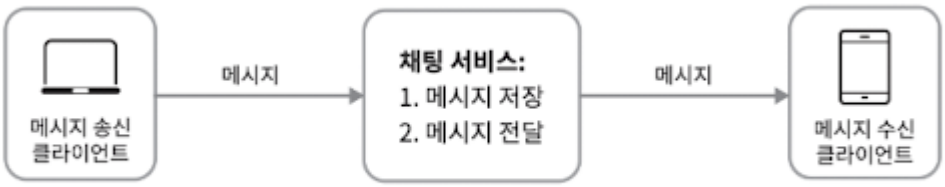


그림 12-2

- 클라이언트가 네트워크 통신 프로토콜을 이용하여 서비스에 접속.
- 송신 클라이언트가 메시지 보낼 때 HTTP 프로토콜 사용.
- 채팅 서비스와의 접속에는 keep-alive 헤더를 사용하면 효율적. 클라이언트와 서버 사이의 연결을 끊지 않고 계속 유지할 수 있고 TCP 의 hand shake 횟수도 줄일 수 있다.
- 하지만 HTTP 는 클라이언트가 연결을 만드는 프로토콜이고 서버에서 클라이언트로 임의 시점에 메시지를 보내는 데 쉽게 쓰일 수 없다. 서버가 연결을 만드는 것처럼 동작할 수 있도록 `폴링`, `롱 폴링`, `웹소켓` 등의 기법이 사용되어져왔다.

폴링

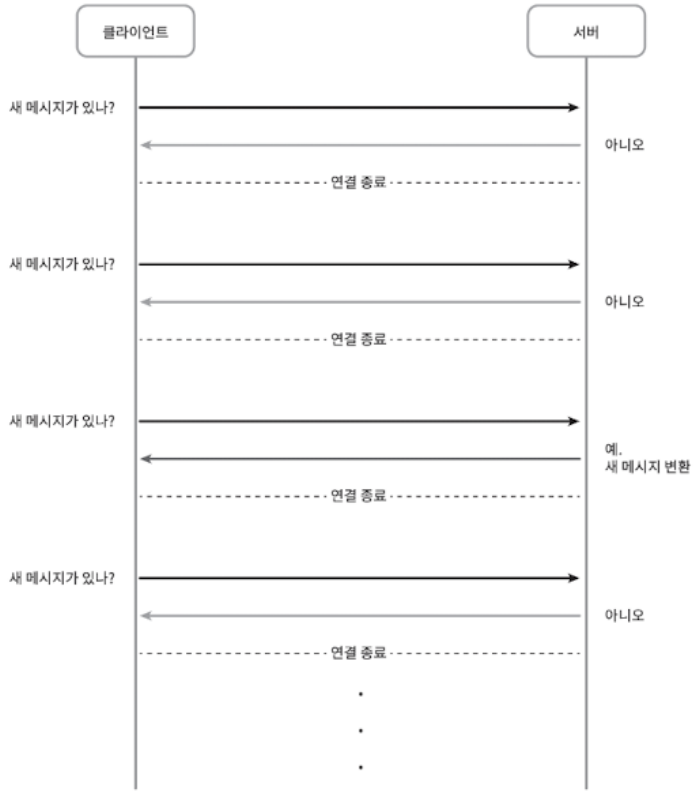


그림 12-3

- 클라이언트가 주기적으로 서버에게 새 메시지가 있는지 물어보는 방법.
- 폴링 자주할수록 비용 증가. 답할 메시지 없으면 서버 자원 낭비되는 문제.

롱 폴링

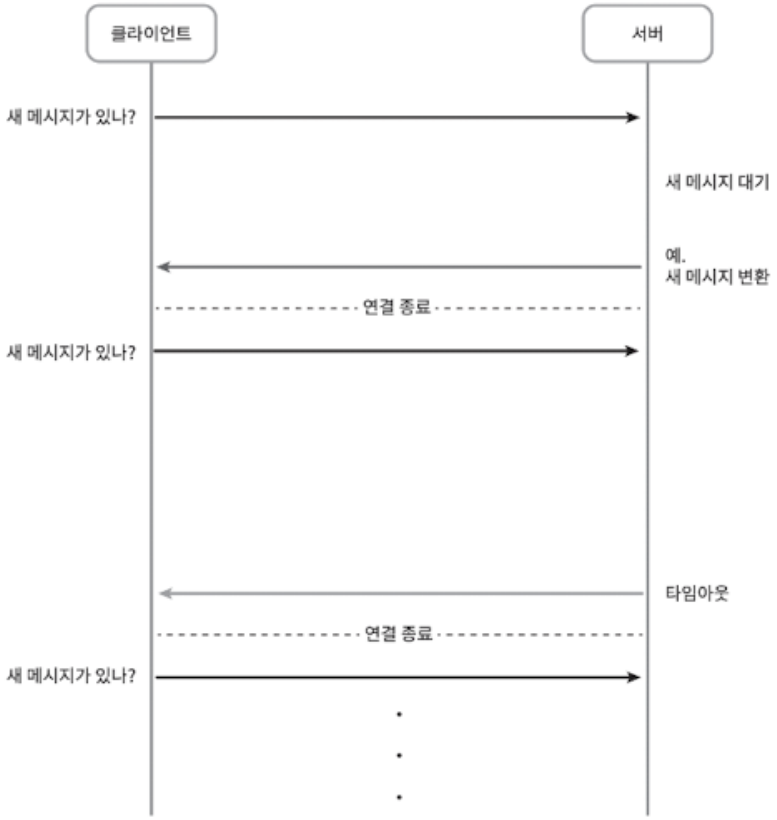


그림 12-4

- 클라이언트는 새 메시지가 반환되거나 타임아웃 될 때까지 연결 유지.
- 새 메시지 받으면 기존 연결 종료. 서버에 새로운 요청 보내어 모두 다시 시작.
- 약점
  - 메시지 보내는 클라이언트와 수신하는 클라이언트가 같은 채팅 서버에 접속하게 되지 않을 수도. HTTP 서버들은 보통 무상태. 로드밸런싱위해 라운드 로빈 사용하는 경우 메시지 받는 서버가 수신 클라이언트와 연결 갖지 않을 수 있다.
  - 서버는 클라이언트가 연결 해제했는지 아닌지 알 좋은 방법이 없다.
  - 비효율적. 메시지 많이 받지 않는 클라이언트도 타임아웃이 일어날 때마다 주기적으로 서버 다시 접속.

웹소켓

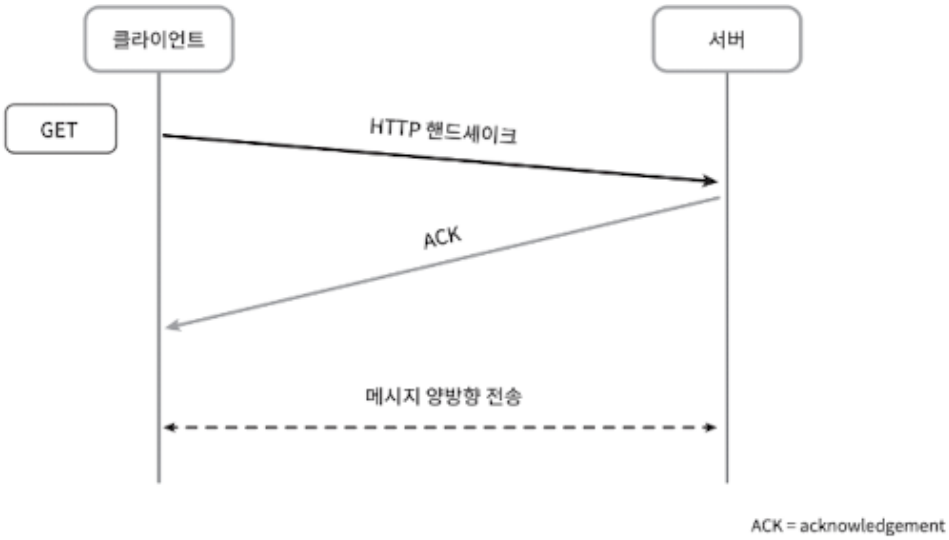


그림 12-5

- 서버가 클라이언트에게 비동기 메시지 보낼 때 가장 많이 사용하는 기술.
- 클라이언트에서 연결 시작. 항구적, 양뱡향.
- 처음에는 HTTP 연결이지만 hand shake 통해 웹소켓 연결로 업그레이드.
- 80, 443 처럼 HTTP, HTTPS 프로토콜이 사용하는 기본 포트 번호 그대로 쓰므로 방화벽 환경에서도 잘 동작.

개략적 설계안

무상태 서비스

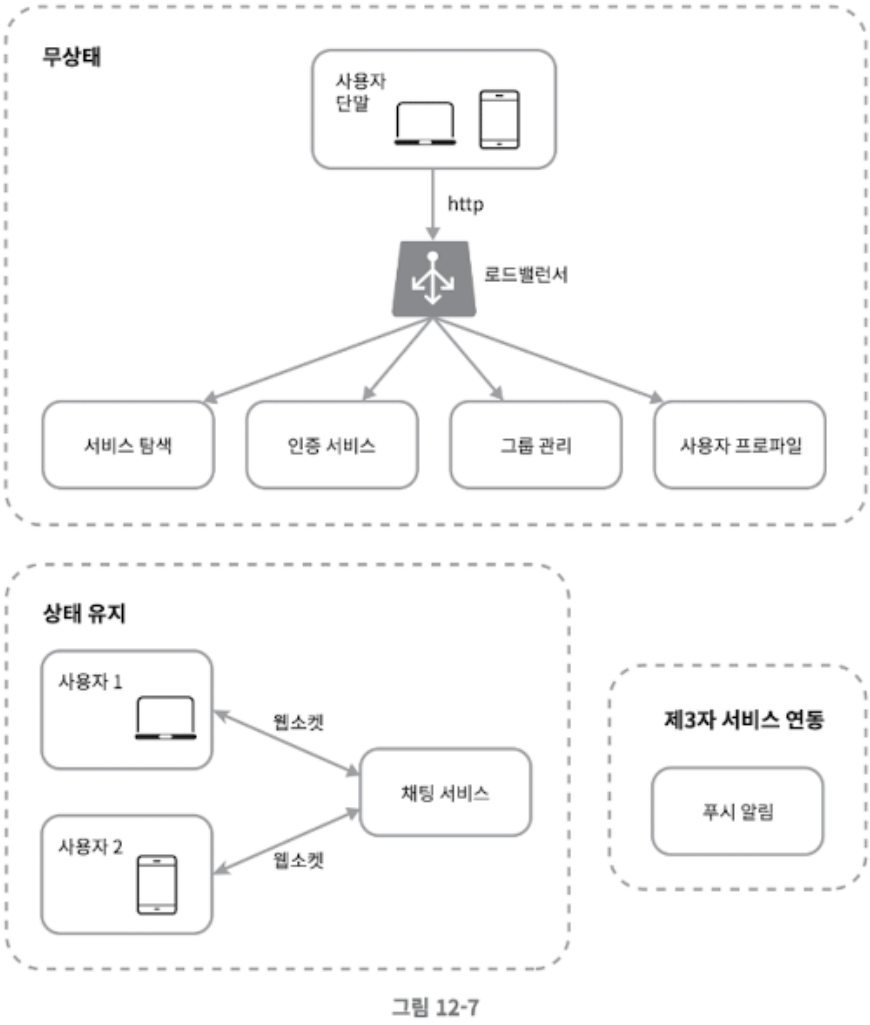


그림 12-7

- 로그인, 회원가입, 사용자 프로필 표시 등을 처리하는 전통적인 요청/응답 서비스.
- 로드밸런서 뒤에 위치. 로드밸런서는 요청을 경로에 맞는 서비스로 전달하고, 그 뒤에 오는 서비스는 모놀리틱일수도, 마이크로서비스일 수도 있다.

상태 유지 서비스

채팅 서비스에서 상태 유지 필요. 클라이언트는 보통 서버가 살아있는 한 다른 서버로 변경하지 않는다. 또한 서비스 탐색 서비스는 채팅 서비스와 협력하여 특정 서버의 부하를 막는다.

제 3자 서비스 연동

가장 중요한 제 3자 서비스는 푸시 알림. 새 메시지 받으면 앱이 실행중이지 않더라도 알림 받아야 하므로.

규모 확장성

이번 시스템에서는 동시 접속자가 1M이라고 가정했을때 접속당 10K의 서버 메모리가 필요하다면 개략적으로 10GB 메모리만 있으면 모든 연결을 다 처리할 수 있다.

하지만 서버 한 대의 설계안으로는 SPOF 문제를 포함해서 여러 가지 이유로 좋은 점수를 받기 어렵다.

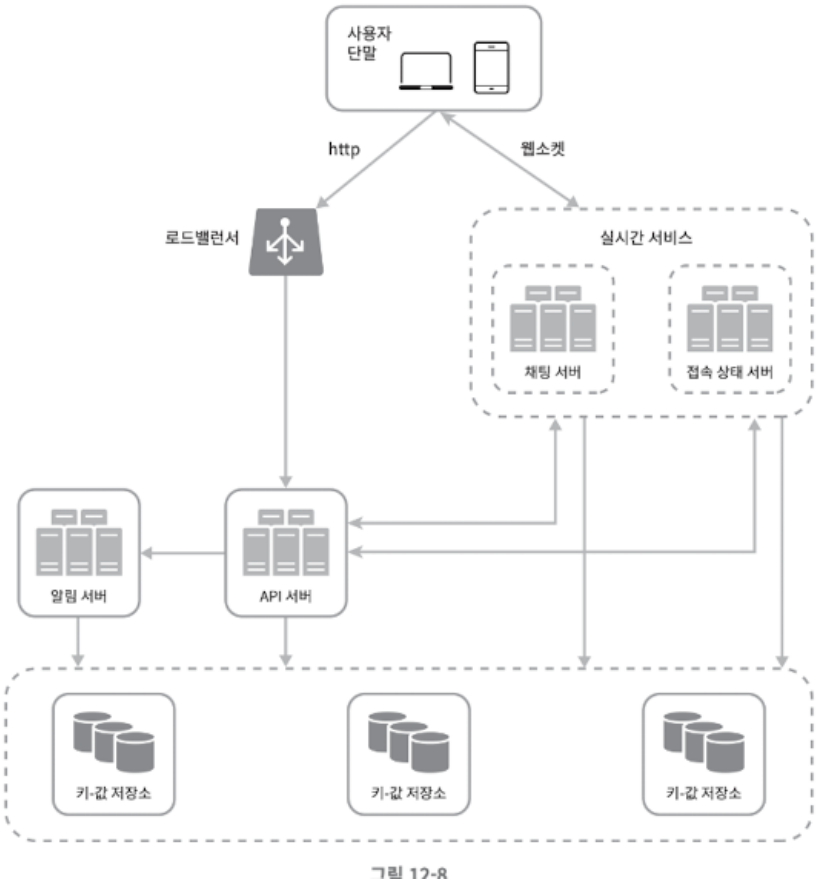


그림 12-8

유의할 것은 실시간 메시지를 위해 클라이언트는 서버와 웹소켓 연결을 유지한다는 것이다.

- 채팅 서버는 클라이언트 사이에 메시지 중계.
- 접속상태 서버는 사용자의 접속 여부 관리.
- API 서버는 로그인, 회원가입, 프로필 변경 등 그 외 나머지 전부 처리.
- 알림 서버는 푸시 알림 보냄.
- 키-값 저장소에는 채팅 이력 보관. 시스템에 접속한 사용자는 이전 채팅 이력 전부 보게 됨. |

저장소

> 데이터 계층 올바르게 쓰려면 어떤 DB를 쓰느냐가 중요. RDBMS를 채택할 것이냐 NoSQL을 쓸것이냐는 데이터의 유형과 읽기/쓰기 연산 패턴을 고려해야 한다.

데이터의 유형은 보통 두 가지이다.

- 사용자 프로필, 설정, 친구 목록처럼 일반적인 데이터. 이런 것들은 안정성 보장하는 RDBMS 에 저장한다.
- 채팅 이력과 같은 채팅 시스템에 고유한 데이터. 이를 어떻게 보관할지 결정하려면 읽기/쓰기 연산 패턴을 이해해야 한다.

읽기/쓰기 연산 패턴

- 채팅 이력 데이터는 규모가 매우 큼. 페이스북 메신저, 왓츠앱은 매일 600억개 처리.
- 이 중 빈번히 사용되는 것은 최근에 주고받은 메시지. 대부분은 오래된 메시지는 안들여다봄.
- 최근 주고받은 메시지 데이터만 보는건 맞지만 검색 기능 이용이나 특정 사용자가 언급된 메시지를 보거나 특정 메시지로 점프하는 등 무작위 데이터 접근도 지원해야.
- 1:1 채팅 앱은 읽기:쓰기 비율이 대략 1:1.

여기서는 키-값 저장소 (NoSQL) 을 채택. 그 이유는 다음과 같다.

- 수평적 규모 확장이 쉽다.
- 데이터 접근 지연시간이 낮다.
- RDBMS 는 데이터 가운데 롱테일에 해당하는 부분을 잘 처리 못하는 경향이 있는데, 인덱스가 커리면 데이터에 대한 무작위적 접근을 처리하는 비용이 늘어난다.
- 이미 많은 안정적 채팅 시스템이 키-값 저장소를 채틱중이다.

데이터 모델

1:1 채팅을 위한 메시지 테이블

- `message_id` 를 기본키로 하고 메시지 순서를 쉽게 정할 수 있도록 하는 역할도 담당.
- created_at을 사용하여는 메시지 순서 정할 수 없음. 동시에 만들어 질 수도 있으므로.

그룹 채팅을 위한 메시지 테이블

- (`channel_id`, `message_id`) 의 복합 키를 기본 키로 사용.
- `channel_id`는파티션 키로도 사용. 그룹 채팅에 적용될 모든 질의는 특정 채널 대상이므로.

메시지 ID

- `message_id` 값은 고유해야.
- ID 값은 정렬 가능해야 하며 시간 순서와 일치해야.

이 조건들을 어떻게 만족시키는가?

- RDBMS라면 auto_increment 가 대안. NoSQL은 해당 기능을 제공하지 않는다.
- 스노플레이크 같은 전역적 64-bit 순서 번호 생성기 이용.
- 지역적 순서 번호 생성기 이용. ID 의 유일성을 같은 그룹안에서만 보장하면 충분하다는 것.

3단계 상세 설계

서비스 탐색

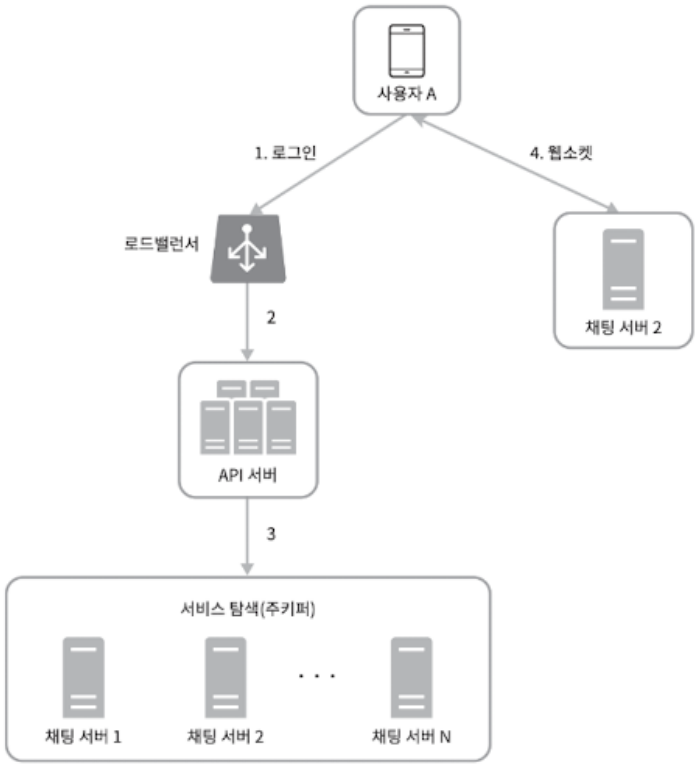


그림 12-11

서버 탐색 기능의 주된 역할은 클라이언트에게 가장 적합한 채팅 서버 추천하는 것. 클라이언트의 위치, 서버의 용량 등을 기준으로 사용. 아파치 주키퍼 같은 오픈 소스 솔루션을 통해 기능 구현.

1. 사용자가 시스템에 로그인 시도
2. 로드 밸런서가 로그인 요청을 API 서버로 보냄.
3. API 서버가 사용자 인증 처리하고 서비스 탐색이 동작하여 해당 사용자를 위한 최적의 채팅 서버를 찾는다. 여기서 채팅 서버 2가 사용자에게 반환되었다고 가정.
4. 사용자는 채팅 서버 2와 웹소켓 연결을 맺는다.

메시지 흐름

1:1 채팅 메시지 처리 흐름

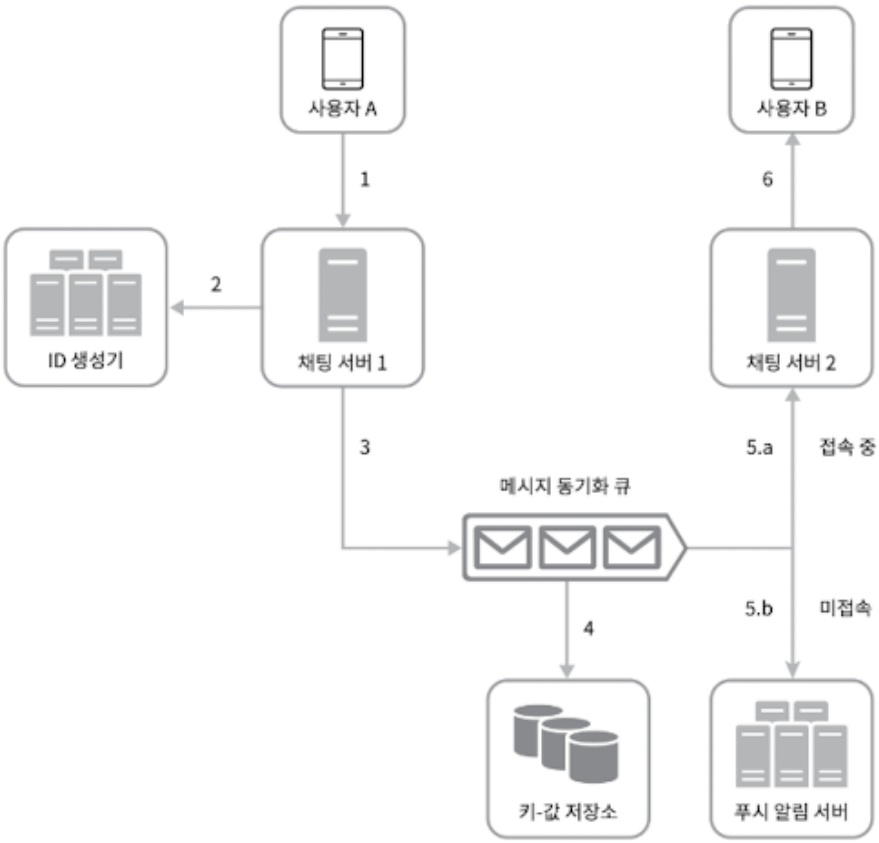


그림 12-12

1. 사용자A가 채팅 서버 1로 메시지 전송.
2. 채팅 서버 1은 id 생성기를 사용해 해당 메시지의 id 결정
3. 채팅 서버 1은 해당 메시지를 메시지 동기화 큐로 전송
4. 메시지가 키-값 저장소에 보관.
5. a) 사용자 B가 접속중이면 메시지는 해당 접속중인 서버 (채팅 서버2)로 전송됨 b) 사용자 B가 접속중이 아니면 푸시 알림 메시지를 푸시 알림 서버로 보냄
6. 채팅 서버2는 메시지를 사용자 B에게 전송, 웹 소켓 연결이 이미 있으므로 그것을 이용.\

여러 단말 사이의 메시지 동기화

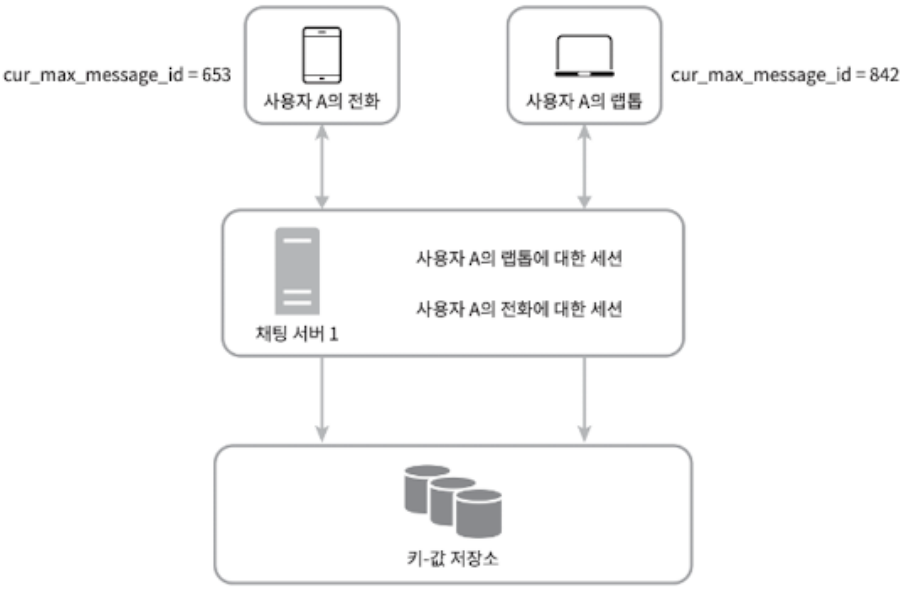


그림 12-13

사용자 A는 전화기와 랩탑 두 대의 단말을 사용 중. 사용자 A가 전화기에서 채팅 앱에 로그인 한 결과로 채팅 서버 1과 해당 단말 사이에 웹 소켓이 만들어진다. 각 단말은 `cur_max_message_id` 변수를 유지. 해당 단말에서 관측된 가장 최신 메시지의 ID를 추적하는 용도. 아래 두 조건을 만족하는 메시지는 새 메시지로 간주한다.

- 수신자 ID가 현재 로그인한 사용자 ID와 같다.
- 키-값 저장소에 보관된 메시지로, 그 ID가 `cur_max_message_id`보다 크다.

`cur_max_message_id`는 단말마다 별도로 유지 관리하면 되는 값이므로 키-값 저장소에서 새 메시지를 가져오는 동기화 작업도 쉽게 구현할 수 있다.

소규모 그룹 채팅에서 메시지의 흐름

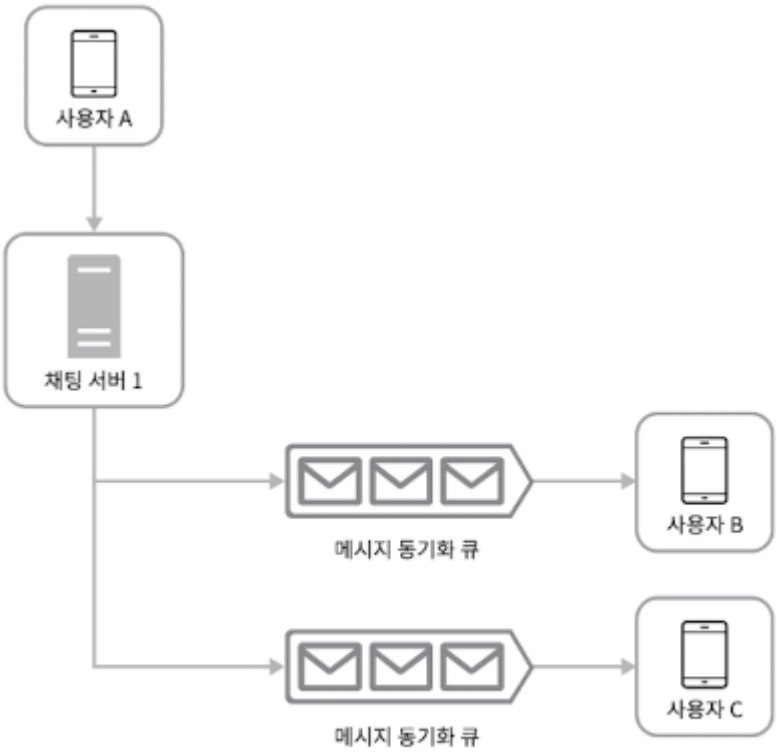


그림 12-14

사용자 A가 그룹 채팅 방에서 메시지를 보내면 메시지가 사용자 B, C앞에 존재하는 각각의 메시지 동기화 큐에 들어가는데, 이 큐는 각 사용자를 위한 메시지 수신함 같은 것.

소규모 채팅방에 적합한 이유

- 새로운 메시지가 왔는지 확인하려면 자기 큐만 보면 되니까 메시지 동기화 플로우가 단순하다.
- 그룹이 크지 않으면 메시지를 수신자별로 복사해서 큐에 넣는 작업의 비용이 문제가 되지 않는다.