2️⃣

231012 12. 채팅 시스템 설계

▼ 문제 이해 및 설계 범위 확정

요구사항

- 모바일, 웹 둘 다 지원
- 5천만 DAU(Daily Active User)를 처리 가능
- 최대 100명까지 참여할 수 있는 그룹 채팅
- 응답 지연이 낮은 일대일 채팅
- 사용자의 접속상태 표시
- 하나의 계정으로 다양한 단말에 동시 접속 가능
- 푸시 알림

▼ 개략적 설계안 제시 및 동의 구하기

Figure 12-2

클라이언트와 채팅 서비스 사이의 관계

기본적으로 지원해야 하는 기능

- 클라이언트들로부터 메시지 수진
- 메시지 수진자 결정 및 전달
- 수신자가 접속 상태가 아닌 경우에는 접속할 때까지 해당 메시지 보관

어떤 통신 프로토콜을 사용할 것인가

- 클라이언트/서버 애플리케이션에서 요청을 보내는 것은 클라이언트 ⇒ 메시지 송신 클라이언트
- 클라이언트는 채팅 서비스에 HTTP 프로토콜로 연결한 다음 메시지를 보내서 수신자에게 해당 메시지들 전달하려고 알림
- keep-alive 헤더
  - 채팅 서비스와의 접속에 사용하면 효율적임
  - 클라이언트와 서버 사이의 연결을 끊지 않고 계속 유지 가능
  - TCP 접속 과정에서 발생하는 핸드셰이크 횟수를 줄일 수 있음

메시지 수신 시나리오

- HTTP는 클라이언트가 연결을 만드는 프로토콜이라 서버에서 클라이언트로 임의 시점에 메시지를 보내는 데는 쉽게 쓰일 수 없음
- 폴링, 롱 폴링, 웹소켓 등의 기술과 기법들로 서버가 연결을 만드는 것처럼 동작할 수 있도록 함

▼ 폴링

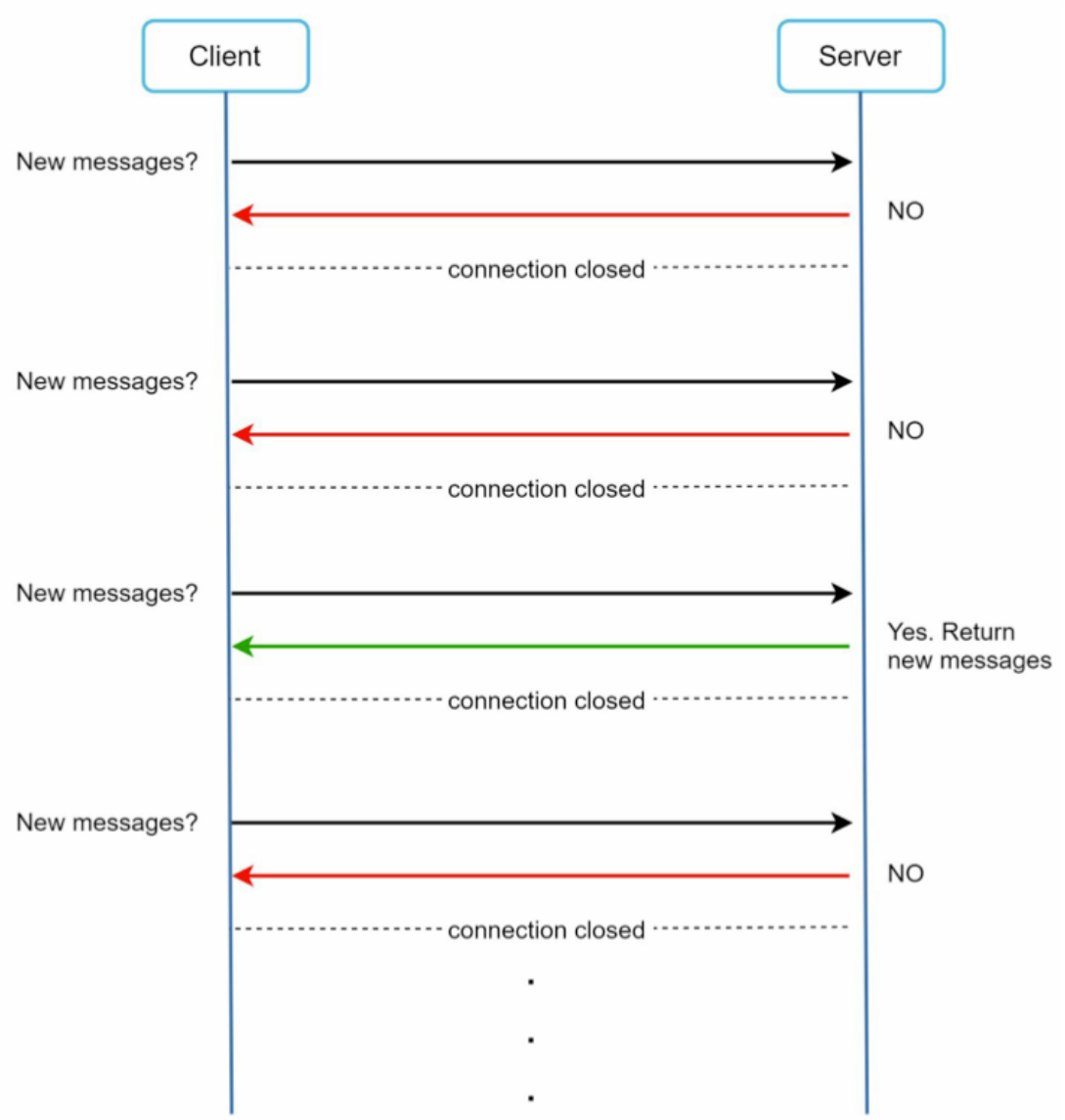


Figure 12-3

- 클라이언트가 주기적으로 서버에세 새 메시지가 있냐고 물어보는 방식
- 폴링 비용
  - 폴링을 자주 할수록 올라감
- 답해줄 메시지가 없는 경우에는 서버 자원이 불필요하게 낭비됨

▼ 롱 폴링

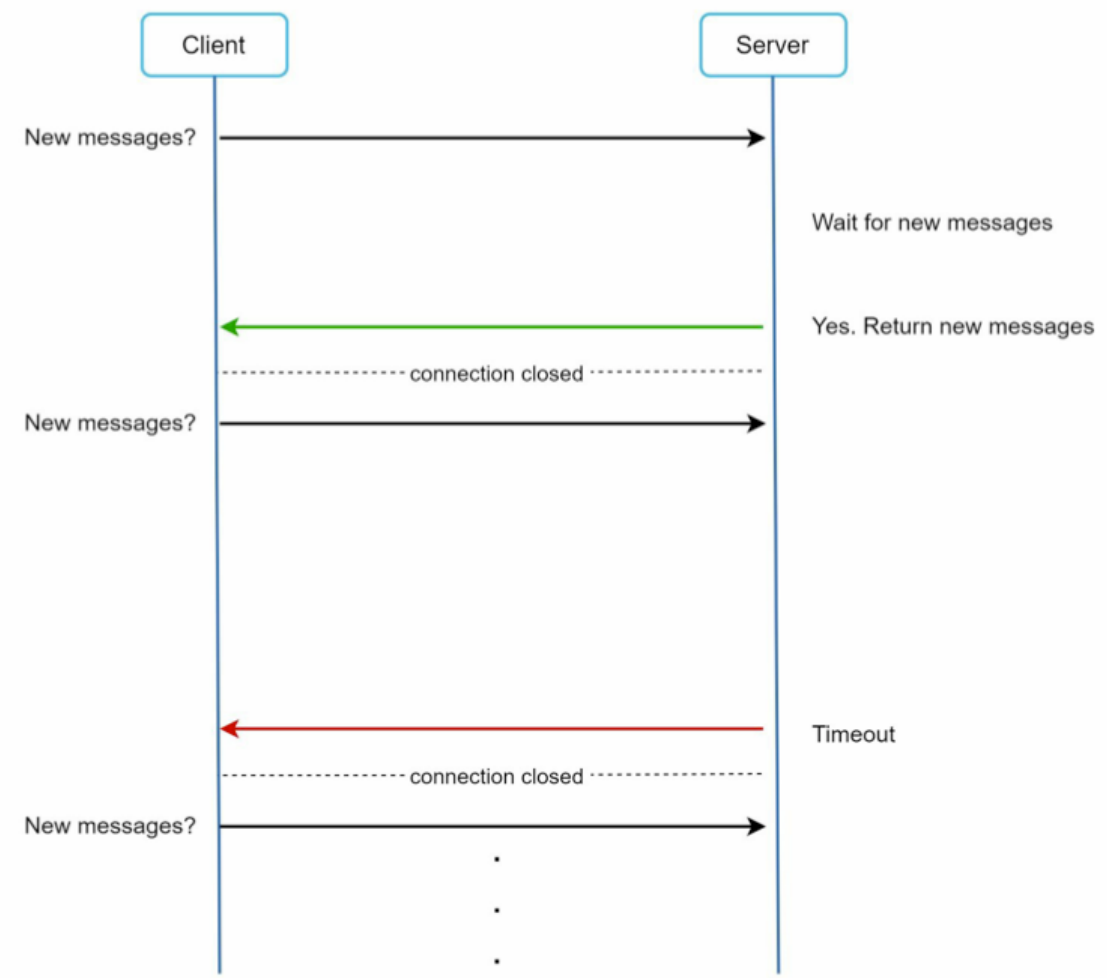


Figure 12-4

- 폴링의 비효율적인 이슈들을 해결한 기법
- 클라이언트는 새 메시지가 반환되거나 타임아웃 될 때까지 연결을 유지
- 새 메시지를 받으면 기존 연결 종료하고 서버에 새로운 요청을 보내 모든 절차를 재시작
- 약점
  - 송신자와 수신자가 같은 채팅 서버에 접속하지 않을 수도 있음
  - HTTP 서버들은 보통 무상태 서버인데 로드밸런싱을 위해 라운드 로빈 알고리즘을 사용하면 메시지를 받은 서버는 해당 메시지를 수신할 클라이언트와의 롱 폴링 연결을 가지고 있지 않은 서버일 수도 있음
  - 서버 이장에서 클라이언트가 연결을 해제했는지 안했는지 알 수 있는 좋은 방법이 없음
  - 여전히 비효율적

    - 메시지를 많이 받지 않은 클라이언트도 타임아웃이 일어날 때마다 주기적으로 서버에 다시 접속함

▼ 웹소켓

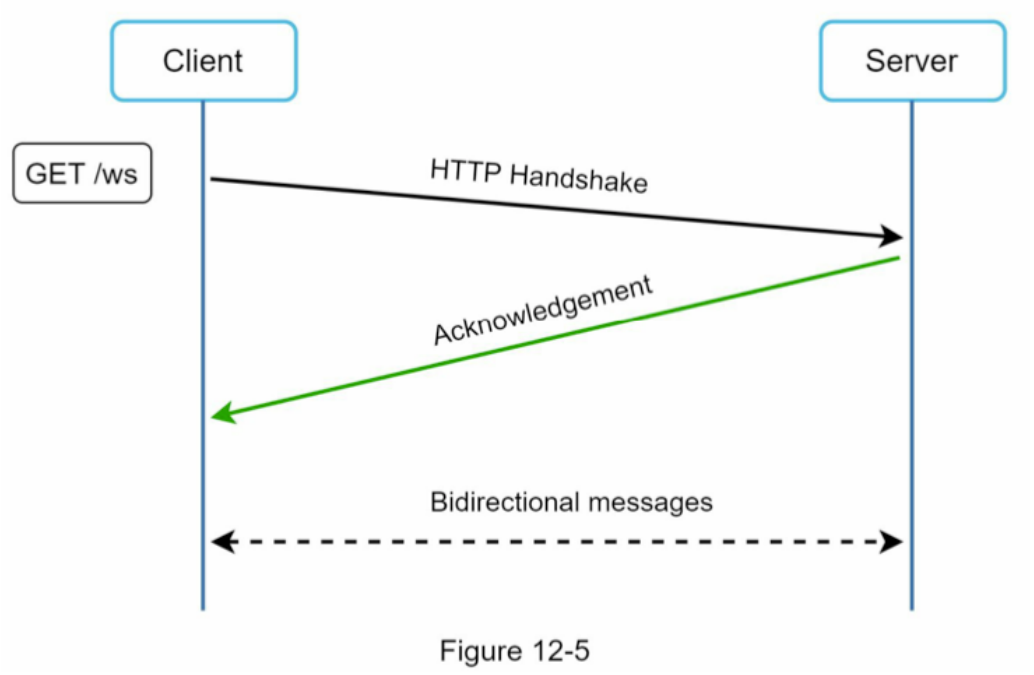


Figure 12-5

- 서버가 클라이언트에게 비동기 메시지를 보낼 때 가장 널리 사용하는 기술
- 웹소켓 연결은 클라이언트가 시작
- 한번 맺어진 연결은 항구적이며 양방향
- 처음에는 HTTP 연결이지만 특정 핸드셰이크 절차를 거쳐 웹소켓 연결로 업그레이드
- 항구적인 연결이 만들어지고 나면 서버는 클라이언트에게 비동기적으로 메시지 전송 가능
- 80이나 443처럼 HTTP, HTTPS 프로토콜이 사용하는 기본 포트를 사용하기 때문에 방화벽이 있는 환경에서도 잘 동작

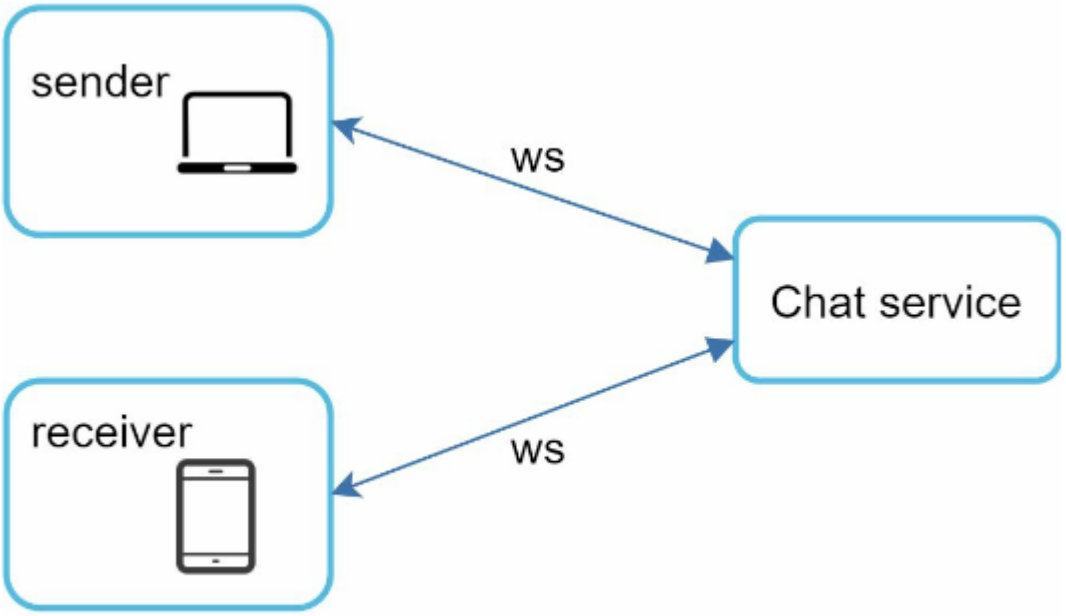
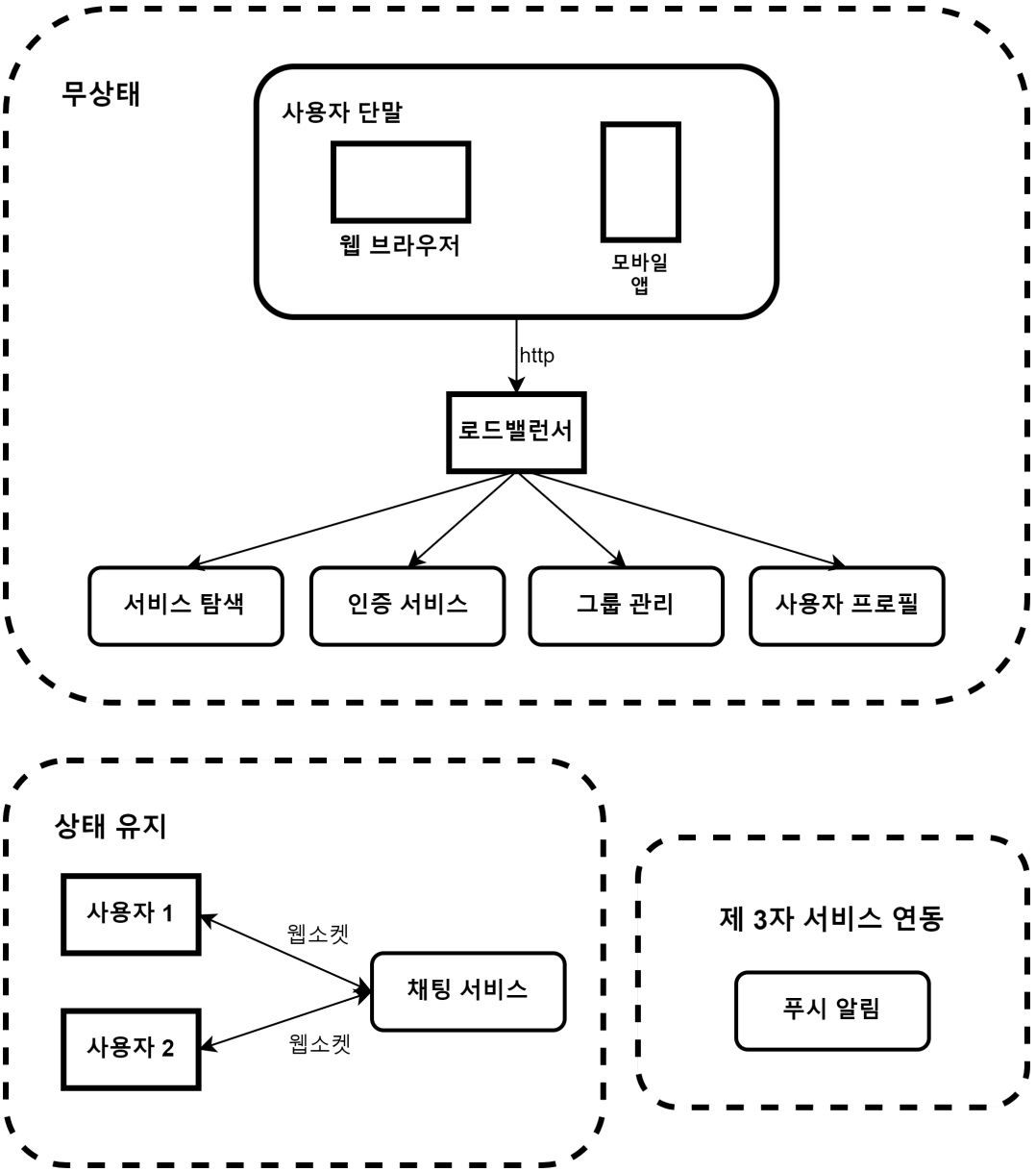


Figure 12-6

- 메시지를 보내고 받을 때 동일한 프로토콜 사용 가능
- 구현도 단순하고 직관적
- 유의할 점
    - 웹소켓 연결은 항구적으로 유지되어야 하기 때문에 서버 측에서 연결 관리를 효율적으로 해야 함

▼ 개략적 설계안



무상태 서비스

- 로그인, 회원가입 등을 처리하는 전동적인 요청/응답 서비스
- 로드밸런서 뒤에 위치
  - 로드밸런서는 요청을 그 경로에 맞는 서비스로 정확하게 전달하는 역할
- 서비스 탐색 서비스
  - 클라이언트가 접속할 채팅 서버의 DNS 호스트명을 클라이언트에게 알려주는 역할

상태 유지 서비스

- 설계안에서 유일하게 상태 유지가 필요한 서비스 ⇒ 채팅 서비스
- 클라이언트가 채팅 서버와 독립적인 네트워크 연결을 유지해야 함
    - 서버가 살아 있는 한 다른 서버로 연결을 변경하지 않음
- 서비스 탐색 서비스는 채팅 서비스와 긴밀하게 협력하여 특정 서버에 부하가 몰리지 않도록 함

제 3자 서비스 연동

- 푸시 알림
- 앱이 실행 중이지 않더라도 알림을 받아야 함

규모 확장성

- 트래픽 규모가 작은 경우 모든 기능을 서버 한 대로 구현 가능
- **서버 한 대로 얼마나 많은 접속을 동시에 허용할 수 있는지**를 따져봐야 함
- SPOF 문제 무조건 발생
- 서버 한 대를 갖는 설계안에서 출발하여 점차 다듬어 나가는 것은 괜찮음 ⇒ **이것은 시작일 뿐**

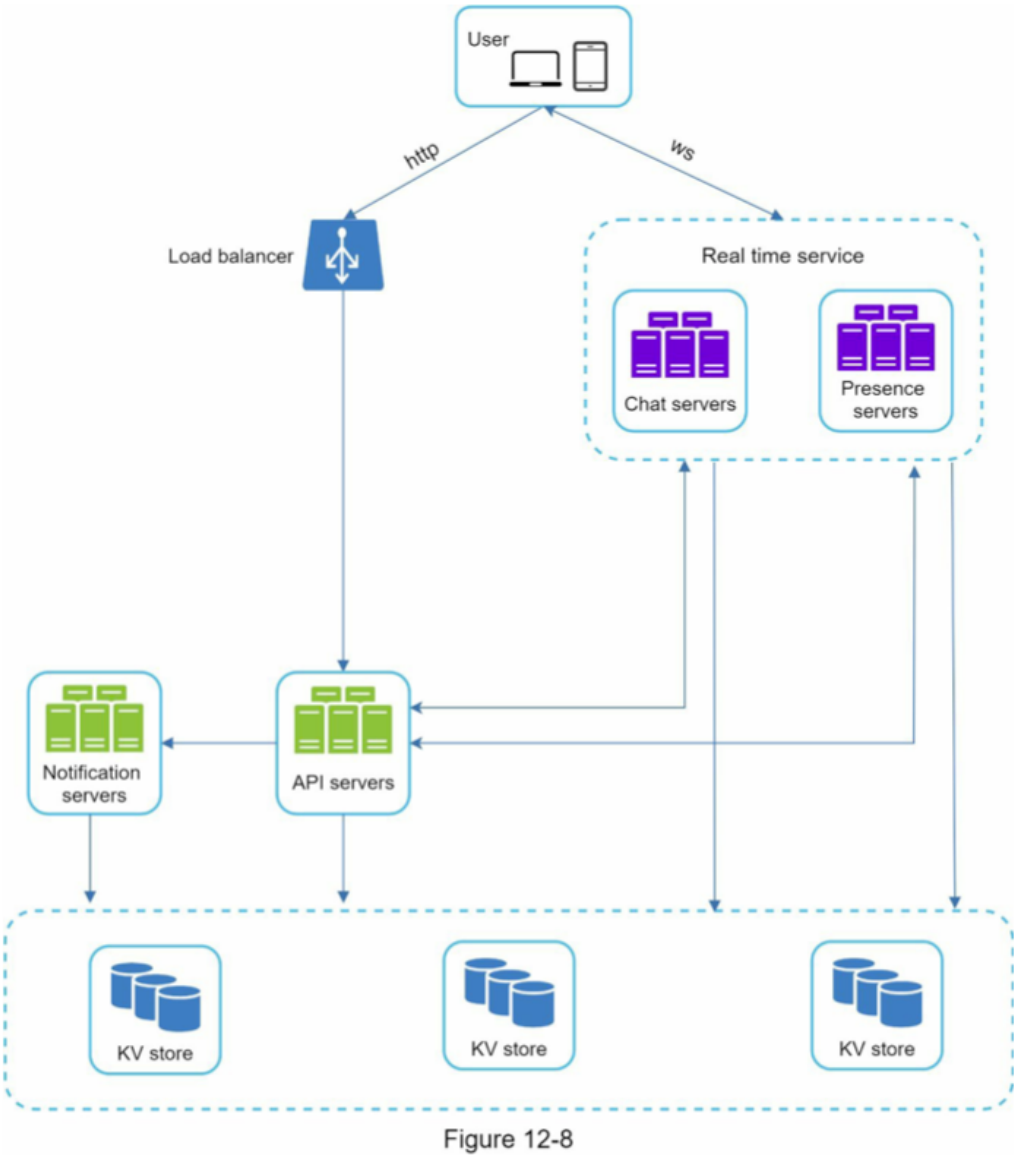


Figure 12-8

- 채팅 서버는 클라이언트 사이에 메시지를 중계하는 역할
- 접속상태 서버는 사용자의 접속 여부를 관리
- API 서버는 로그인, 회원가입 등 그 외 나머지 전부를 처리
- 알림 서버는 푸시 알림
- 키-값 저장소에는 채팅 이력 보관
  - 시스템에 접속한 사용자는 이전 채팅 이력 전부 열람 가능

저장소

- 데이터 계층을 올바르게 만들기 위해서는 **어떤 DB를 쓰는지**가 중요
- 어떤 DB를 사용할 것인지는 **데이터의 유형과 읽기/쓰기 연산의 패턴**이 중요

- 채팅 시스템에서 다루는 데이터
  - 사용자 프로필, 설정, 친구 목록 등 일반 데이터
    - 데이터 안정성을 보장하는 관계형 데이터베이스에 보관
    - 다중화와 샤딩을 사용하여 가용성 및 규모확장성 보증
  - 채팅 이력
    - 읽기/쓰기 연산 패턴을 이해해야 함
    - 채팅 이력 데이터의 양이 많음
    - 이 데이터에 가운데 빈번하게 사용되는 것은 최근에 주고받은 메시지
    - 검색 기능, 특정 사용자 언급 기능, 특정 메시지로 점프 기능처럼 무작위적인 데이터 접근을 할 수도 있음
    - 읽기:쓰기 비율이 1:1

    ⇒ 키-값 저장소를 추천함

      - 수평적 규모확장이 쉬움
      - 데이터 접근 지연시간이 낮음
      - 관계형 DB는 롱 테일에 해당하는 부분을 잘 처리하지 못하는 경향이 있음
      - 이미 많은 안정적인 채팅 시스템이 사용 중

▼ 데이터 모델

1:1 채팅을 위한 메시지 테이블

message	
message_id	**bigint**
message_from	bigint
message_to	bitint
content	text
created_at	timestamp

Figure 12-9

그룹 채팅을 위한 메시지 테이블

Figure 12-10

message_id

- 조건
  - 고유값
  - 정렬이 가능해야 하며 순서와 일치해야 함
- RDBMS
  - auto_increment 사용
- 스노플레이크
- 지역적 순서 번호 생성기
  - ID의 유일성은 같은 그룹 안에서만 보증하면 충분
  - 메시지 사이의 순서는 같은 채널, 같은 1:1 세션 안에서만 유지되면 됨
  - 전역적 ID 생성기에 비해 구현하기 쉬운 접근법

▼ 상세 설계

서비스 탐색

- 클라이언트에게 가장 적합한 채팅 서버를 추천하는 것
- 클라이언트의 위치, 서버 용량을 기준으로 추천

- 사용 가능한 모든 채팅 서버를 등록하고 클라이언트가 접속 시도하면 기준에 따라 최적의 채팅 서버를 골라 줌
- 주키퍼로 구현한 서비스 탐색 기능 동작법

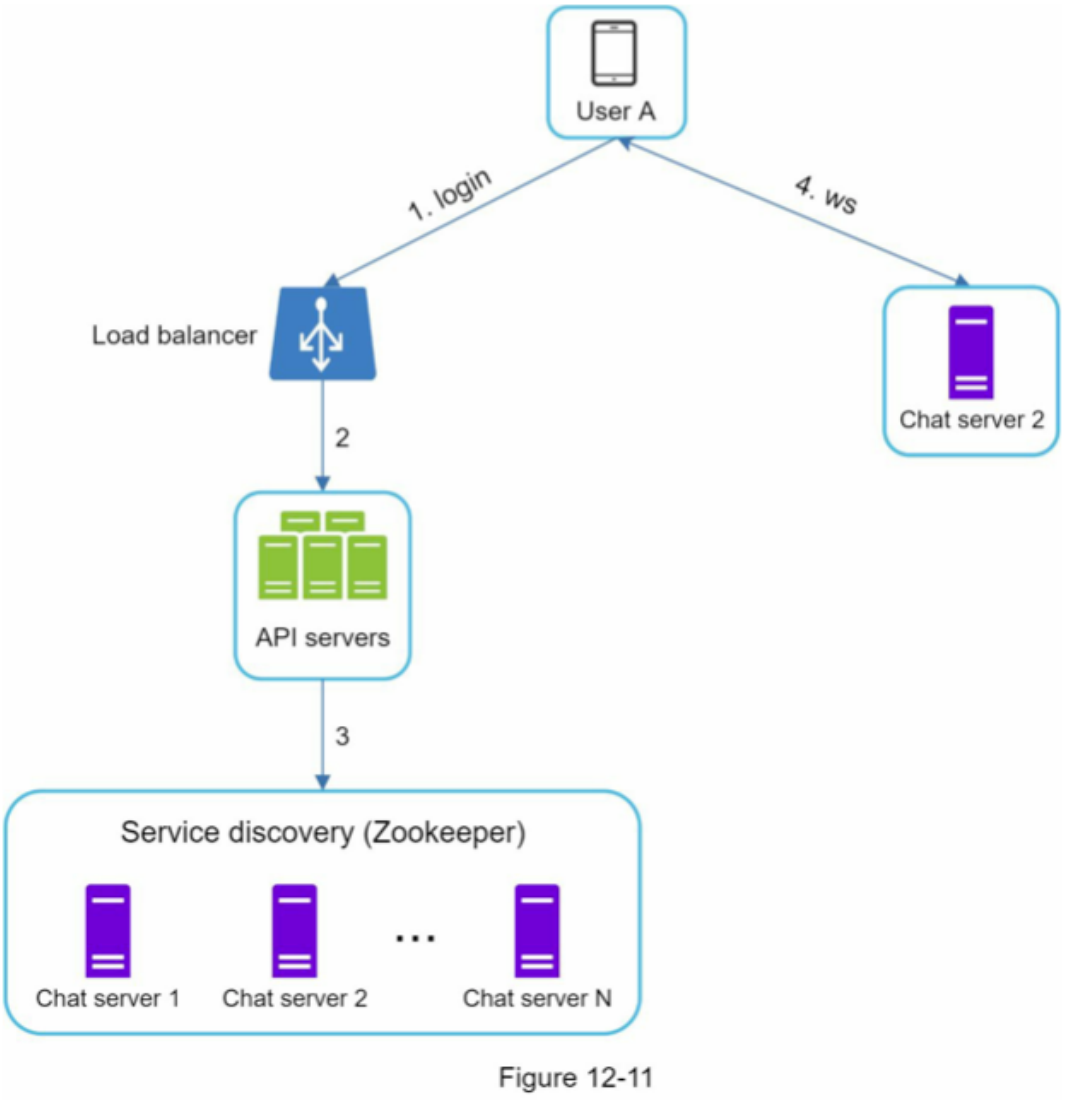


Figure 12-11

1. 사용자가 시스템에 로그인 시도
2. 로드밸런서가 로그인 요청을 API 서버들 가운데 하나로 보냄
3. API 서버가 사용자 인증을 처리하면 서비스 탐색 기능이 동작해 해당 사용자를 서비스할 최적의 채팅 서버를 찾음
4. 사용자는 해당 서버와 웹소켓 연결을 맺음

▼ 메세지 흐름

1:1 채팅 메시지 처리 흐름

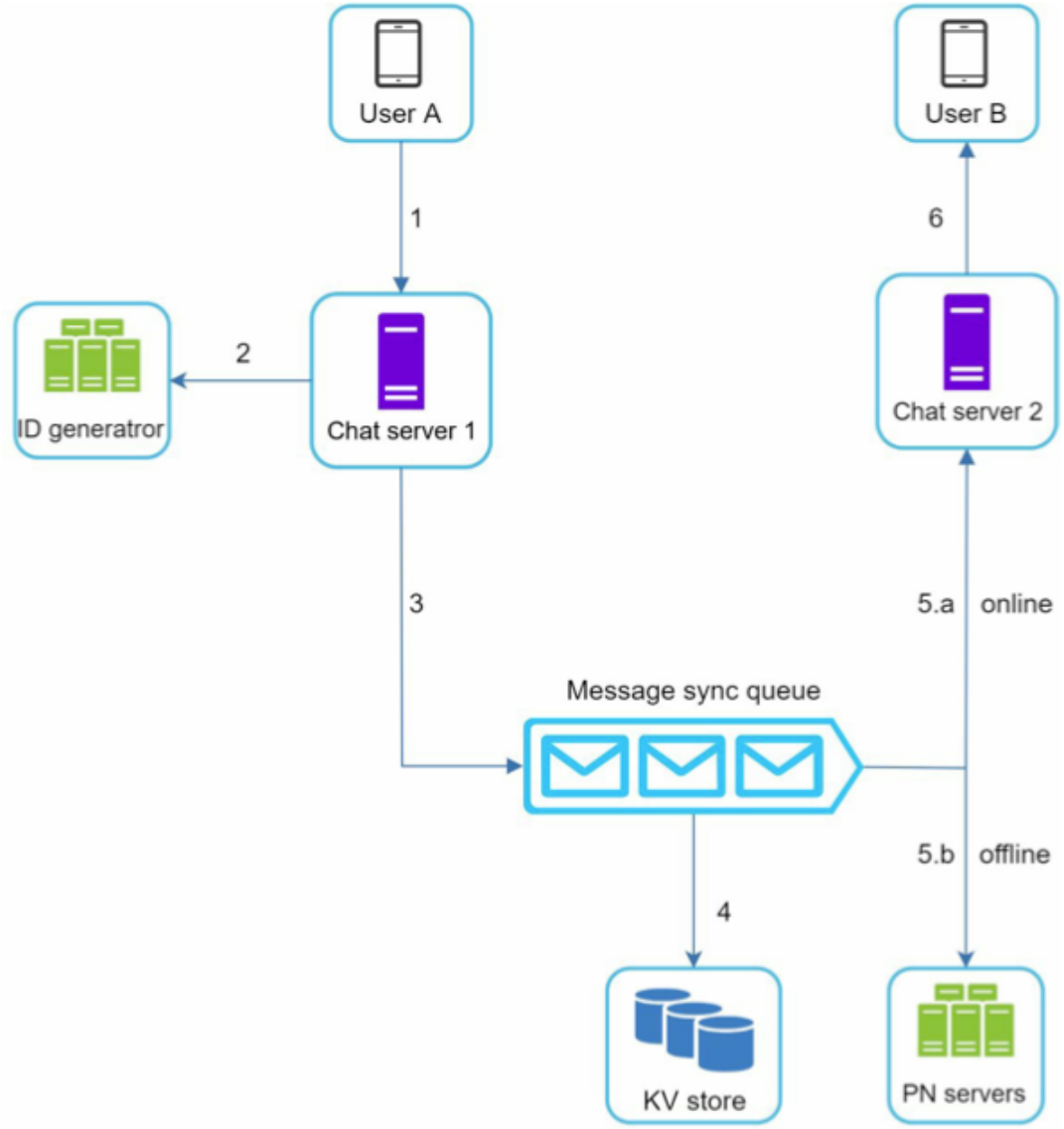


Figure 12-12

1. 사용자가 채팅 서버로 메시지 전송
2. 채팅 서버는 ID 생성기를 사용해 해당 메시지의 ID 결정
3. 채팅 서버는 해당 메시지를 메시지 동기화 큐로 전송
4. 메시지가 키-값 저장소에 보관됨
5. 사용자 B가 접속 중인 경우 메시지는 사용자 B가 접속 중인 채팅 서버 2로 전송됨
6. 사용자 B가 접속 중이 아니라면 푸시 알림 메시지를 푸시 알림 서버로 보냄
7. 채팅 서버 2는 메시지를 사용자 B에게 전송

8. 사용자 B와 채팅 서버 2 사이에는 웹소켓 연결이 있는 상태

여러 단말 사이의 메시지 동기화

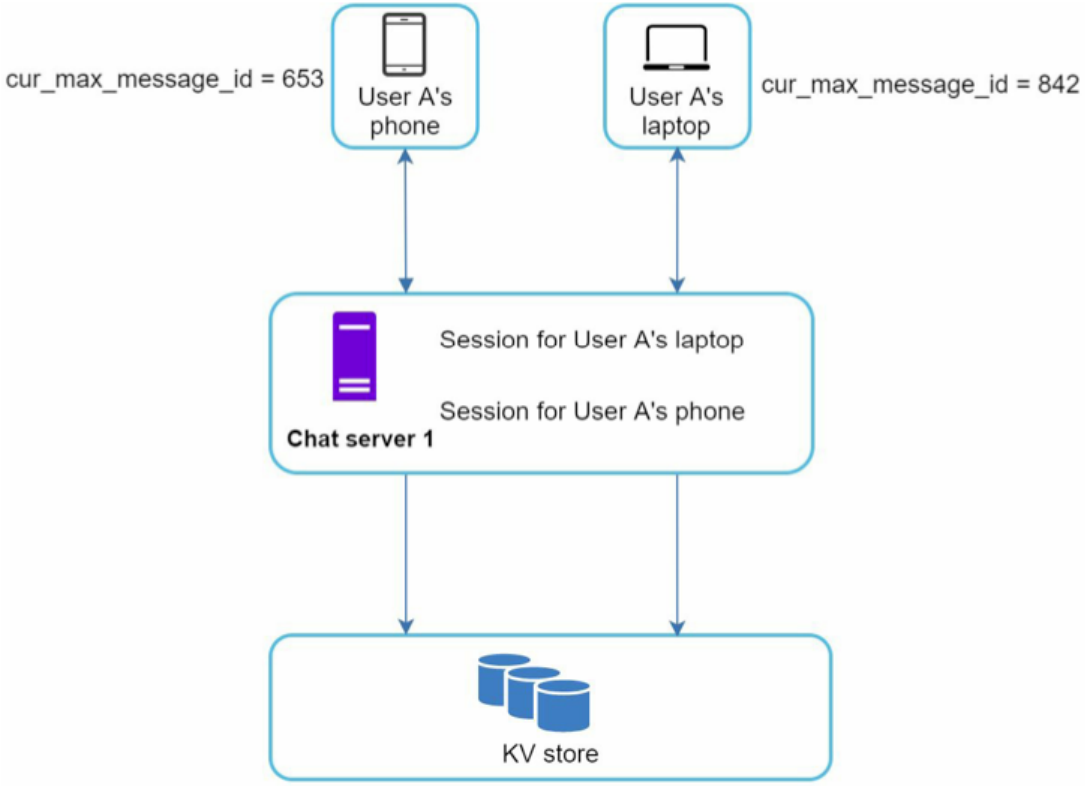


Figure 12-13

- 각 단말은 cur_max_message_id라는 변수를 유지
  - 해당 단말에서 관측된 가장 최신 메시지의 ID를 추적하는 용도
- 새 메시지로 간주하는 조건
  - 수신자 ID가 현재 로그인한 사용자 ID와 같음
  - 키-값 저장소에 보관된 메시지로 그 ID가 cur_max_message_id보다 큼
- cur_max_message_id는 단말마다 별도로 유지 관리하는 되는 값이기 때문에 키-값 저장소에서 새 메시지를 가져오는 동기화 작업도 쉽게 구현할 수 있음

소규모 그룹 채팅에서의 메시지 흐름

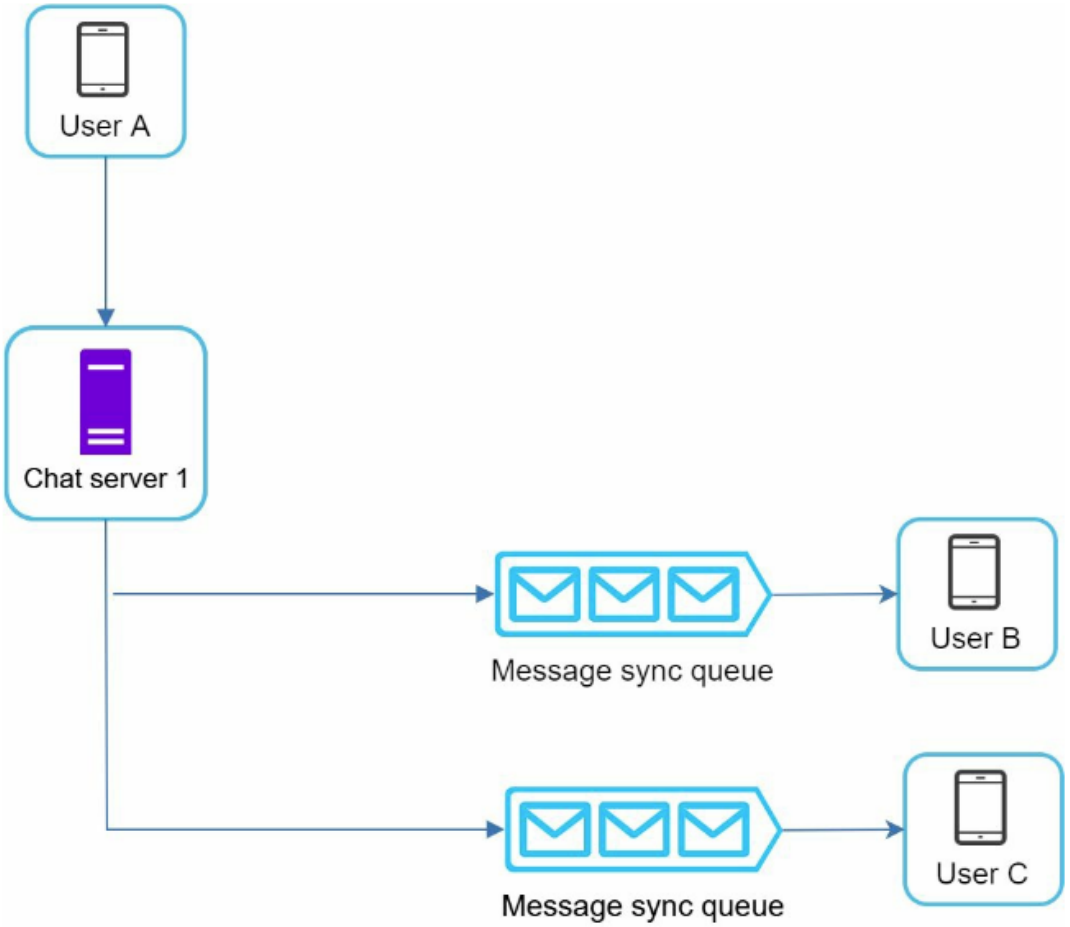


Figure 12-14

- 동작법
  1. 사용자 A가 메시지를 보냄
  2. 사용자 B와 C의 메시지 동기화 큐에 해당 메시지가 복사됨
     - 사용자 각각에 할당된 메시지 수신함 같은 역할
- 새로운 메시지가 왔는지 여부는 내 큐만 보면 됨 ⇒ 메시지 동기화 플로우가 단순
- 그룹이 크지 않으면 메시지를 수신자별로 복사해서 큐에 넣는 작업의 비용이 작음
- 위챗이 사용하고 있음
  - 그룹 크기는 500명으로 제한
- 많은 사용자를 지원해야 하는 경우라면 똑같은 메시지를 모든 사용자의 큐에 복사하는 게 바람직하지 않음

- 수신자 관점에서 보면 한 수신자는 여러 사용자로부터 오늘 메시지를 수신할 수 있어야 함
    - 메시지 동기화 큐는 여러 사용자로부터 오는 메시지를 받을 수 있어야 함

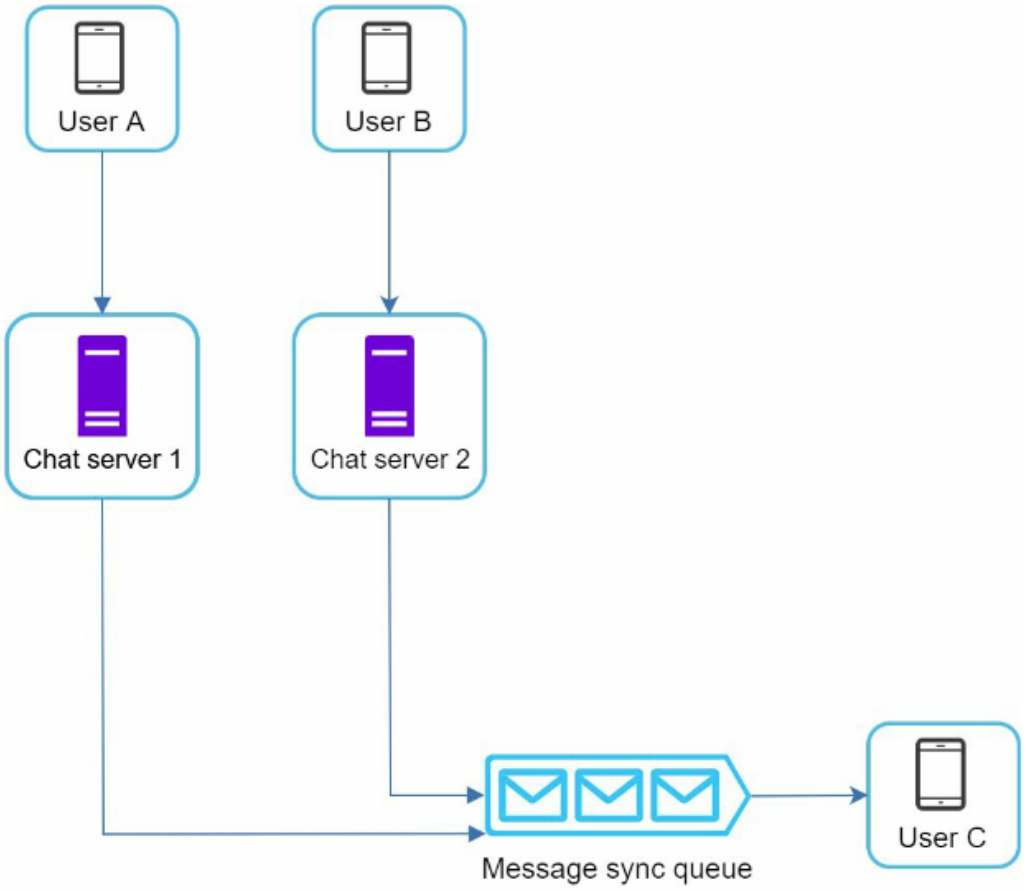


Figure 12-15

▼ 접속상태 표시

채팅 어플리케이션의 핵심적 기능

접속상태 서버는 클라이언트와 웹소켓으로 통신하는 실시간 서비스의 일부라는 점을 유의

사용자 로그인

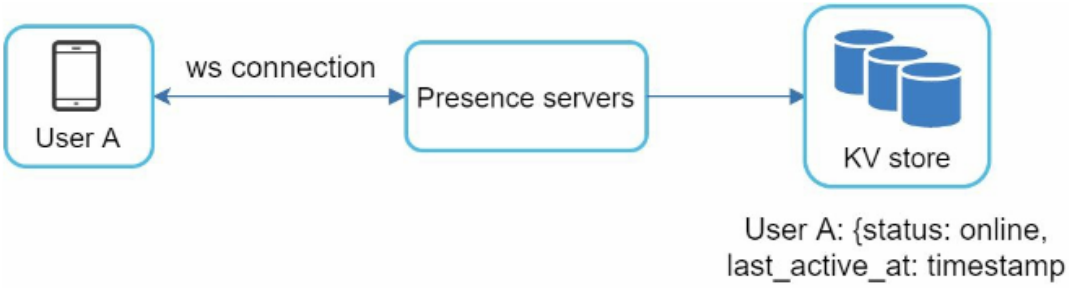


Figure 12-16

- 클라이언트와 실시간 서비스 사이에 웹소켓 연결이 맺어지고 나면 접속상태 서버는 사용자의 상태와 last_active_at 타임스탬프 값을 키-값 저장소에 보관
- 해당 절차가 끝나면 해당 사용자는 접속 중인 것으로 표시

로그아웃

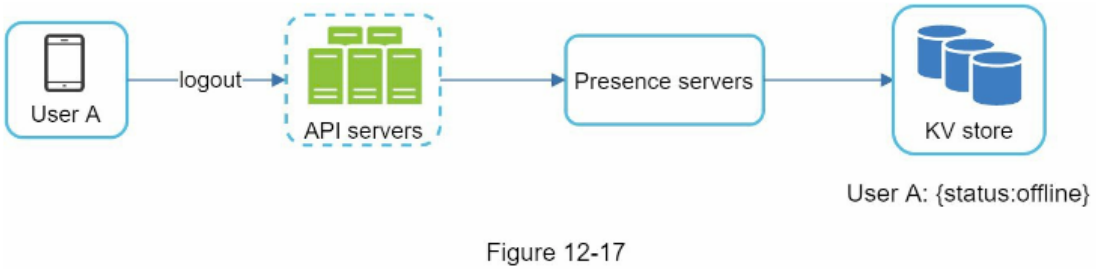


Figure 12-17

- 키-값 저장소에 보관된 사용자 상태가 online에서 offline으로 변경됨

접속 장애

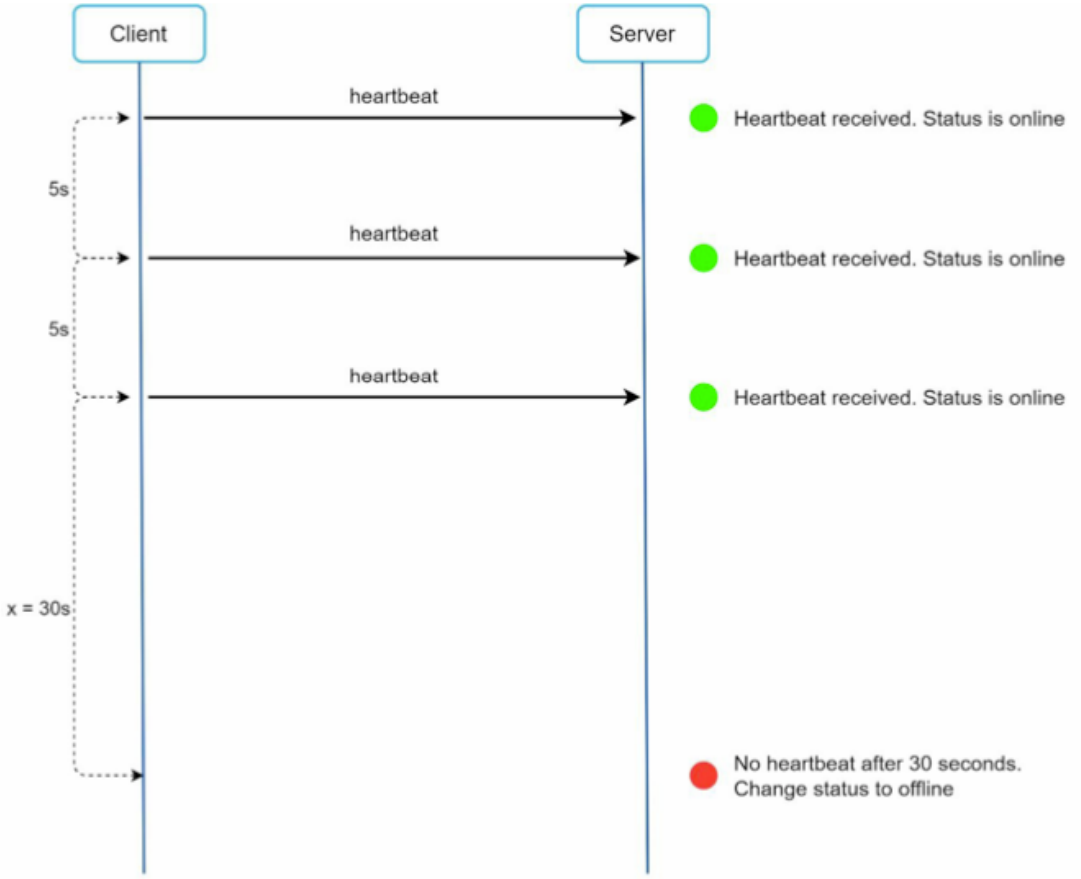


Figure 12-18

- 인터넷을 통한 연결은 항상 안정적이지 못함
- 인터넷 연결이 끊어지면 클라이언트와 서버 사이에 맺어진 웹소켓 같은 지속성 연결도 끊어짐
- 가장 쉬운 대응법
  - 사용자를 오프라인 상태로 변경
  - 연결이 복구되면 온라인 상태로 변경
  - 짧은 시간 동안 인터넷 연결이 끊어졌다 복구되는 일이 많은데 이럴 때마다 사용자의 접속 상태를 변경하는 것은 좋지 못함
- 박동 검사
  - 가장 쉬운 대응법의 단점을 해결한 방법
  - 온라인 상태의 클라이언트로 하여금 주기적으로 박동 이벤트를 접속상태 서버로 보내도록 함
  - 마지막 이벤트를 받은지 n초 이내에 또 다른 박동 이벤트 메시지를 받으면 해당 사용자의 접속상태를 online으로 유지

- 그렇지 않은 경우 offline으로 변경

상태 정보의 전송

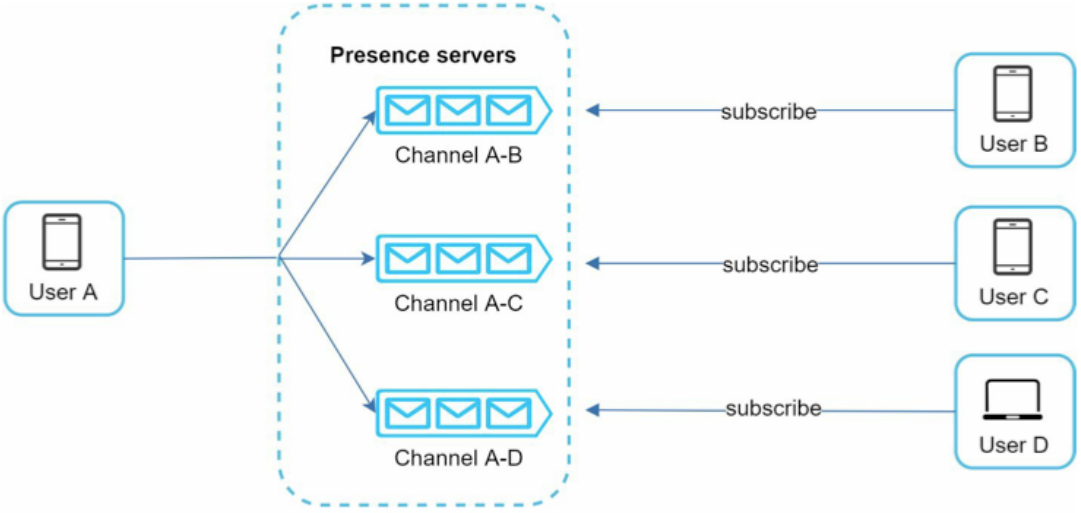


Figure 12-19

- 상태정보 서버는 발행-구독 모델을 사용
- 각각의 친구 관계마다 채널을 하나씩 두고 있음
- 그룹 크기가 작을 때 효과적
- 그룹 크기가 커지면 비용과 시간이 많이 들게 됨
  - 사용자가 그룹 채팅에 입장하는 순간에만 상태 정보를 읽어가게 하기
  - 친구 리스트에 있는 사용자의 접속상태를 갱신하고 싶으면 수동으로 하도록 유도하기

▼ 마무리

채팅 시스템을 설계하면서 추가적으로 고민해 볼 수 있는 사항

- 채팅 앱을 확장하여 사진이나 비디오 등의 미디어를 지원하도록 하는 방법
  - 미디어 파일은 테스트에 비해 크기가 큼
  - 이와 관련하여 압축 방식, 클라우드 저장소, 섬네일 생성 등을 논의
- 종단 간 암호화
  - 왓츠앱은 메시지 전송에 있어 종단 간 암호화를 지원함
  - 메시지 발신인과 수신자 이외에는 아무도 메시지 내용을 볼 수 없다는 뜻

- 캐시
  - 클라이언트에 이미 읽은 메시지를 캐시해 두면 서버와 주고받는 데이터 양을 줄일 수 있음
- 로딩 속도 개선
  - 슬랙은 사용자의 데이터, 채널 등을 지역적으로 분산하는 네트워크를 구축하여 앱 로딩 속도를 개선함
- 오류 처리
  - 채팅 서버 오류
    - 채팅 서버 하나에 수십만 사용자가 접속해 있다고 가정
    - 그런 서버 하나가 죽으며 서비스 탐색 기능이 동작하여 클라이언트에게 새로운 서버를 배정하고 다시 접속할 수 있도록 해야 함
  - 메시지 재전송
    - 재시도나 큐는 메시지의 안정적 전송을 보장하기 위해 흔히 사용되는 기법

▼ 토론

카카오톡처럼 채팅방을 나가기(삭제)가 가능한 경우라면 1:1 채팅에서도 channel_id가 필요하지 않나?

채팅 리스트에 보여질 id들을 가지고 있어야 하니까..