


12장. 채팅 시스템 설계

🕒 Created	@November 17, 2022 12:06 PM
📌 Progress	In Progress


 학습 TODO list


☐


- 8.1. 1단계: 문제 이해 및 설계 범위 확정
- 8.2. 2단계: 개략적 설계안 제시 및 동의 구하기
 - 8.2.1. 클라이언트와 서버의 통신 방법
 - 8.2.2. 폴링
 - 8.2.3. 롱 폴링
 - 8.2.4. 웹소켓
 - 8.2.5. 개략적 설계안
 - 8.2.6. 데이터 모델
- 8.3. 3단계: 상세 설계
 - 8.3.1. 서비스 탐색
 - 8.3.2. 메시지 흐름
 - 8.3.3. 접속상태 표시
- 8.4. 4단계: 마무리


8.1. 1단계: 문제 이해 및 설계 범위 확정


| 질문을 통해 요구사항을 알아내고 설계 범위를 좁힌다.


 어떤 앱을 설계해야 하나요? 1:1 채팅 앱입니까 아니면 그룹 채팅 앱입니까?


 둘 다 지원할 수 있어야 합니다.


 모바일 앱인가요 아니면 웹 앱인가요?

 둘 다입니다.

 처리해야 하는 트래픽 규모는 어느 정도입니까?

 일별 능동 사용자 수(DAU: Daily Active User) 기준으로 5천만(50 million) 명을 처리할 수 있어야 합니다.

 그룹 채팅의 경우에 인원 제한이 있습니까?

 최대 100명까지 참가할 수 있습니다.



중요 기능으로는 어떤 것이 있을까요? 가령, 첨부파일도 지원할 수 있어야 하나요?



1:1 채팅, 그룹 채팅, 사용자 접속상태 표시를 지원해야 합니다. 텍스트 메시지만 주고받을 수 있습니다.



메시지 길이에 제한이 있나요?



네. 100,000자 이하여야 합니다.



종단 간 암호화(end-to-end encryption)를 지원해야 하나요?



현재로서는 필요없습니다만 시간이 허락하면 논의해볼 수 있겠습니다.



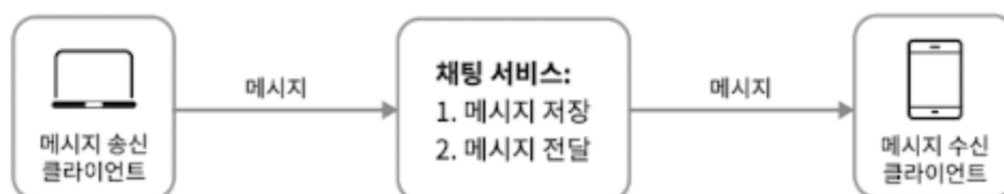
채팅 이력은 얼마나 오래 보관해야 할까요?



영원히요.

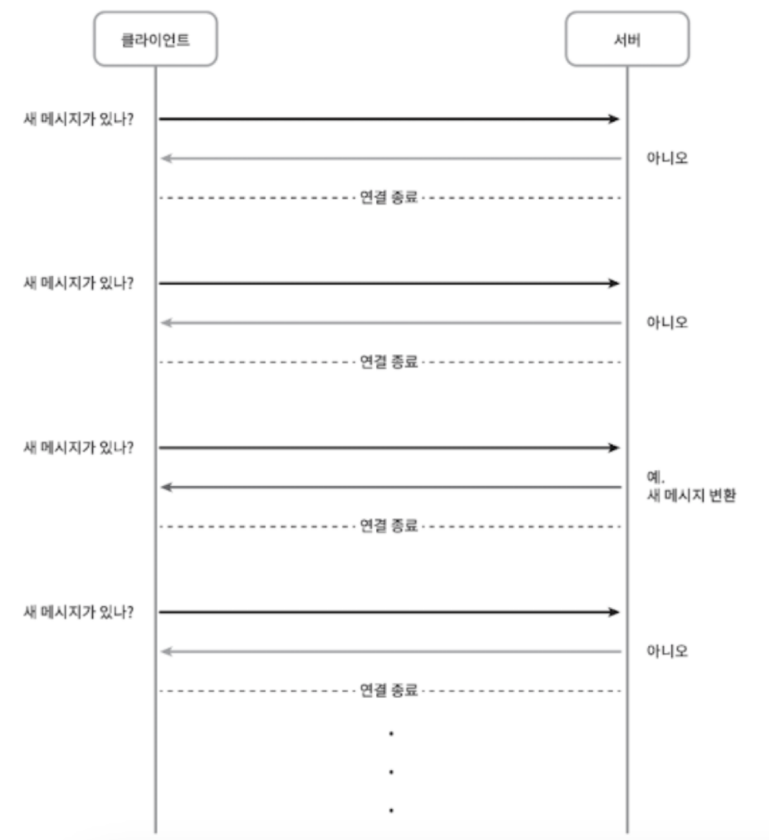
8.2. 2단계: 개략적 설계안 제시 및 동의 구하기

8.2.1. 클라이언트와 서버의 통신 방법



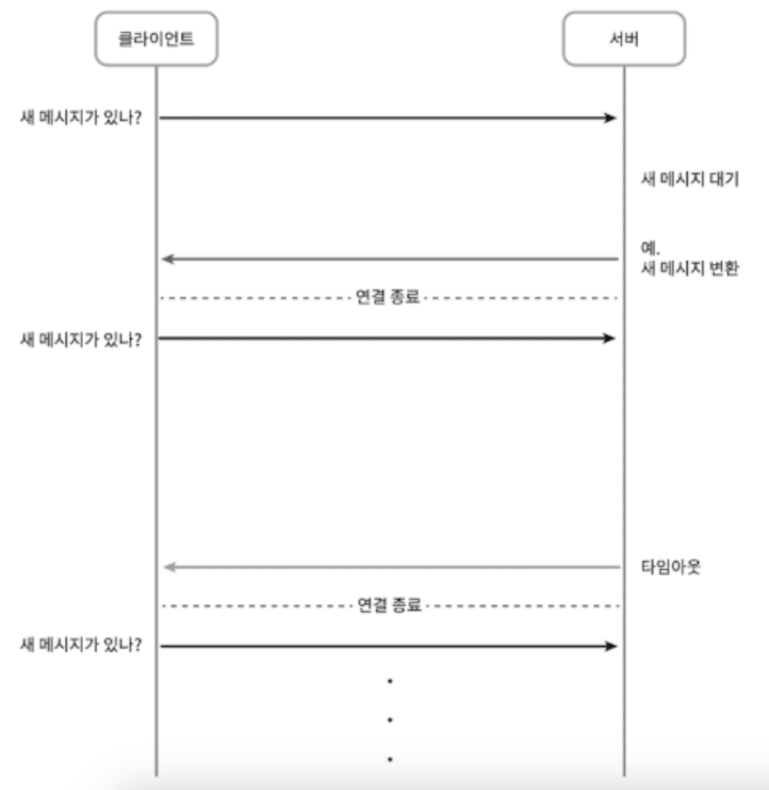
- 채팅을 시작하려는 클라이언트가 네트워크 통신 프로토콜을 사용하여 서비스에 접속
 - 메시지 송신 클라이언트(sender)가 HTTP 프로토콜을 사용하여 메시지를 채팅 서비스에 보내어 수신자에게 메시지를 전달하라고 알린다.
 - 접속에 keep-alive 헤더를 사용하면 효율적이다.
 - 클라이언트와 서버 사이의 연결을 끊지 않고 계속 유지할 수 있다.
 - TCP 접속 과정에서 발생하는 핸드셰이크(handshake) 횟수를 줄일 수 있다.
 - 하지만 HTTP는 클라이언트가 연결을 만드는 프로토콜이므로 서버에서 클라이언트로 임의 시점에 메시지를 보내는데 쉽게 쓰일 수 없다.
 - 서버가 연결을 만드는 것처럼 동작할 수 있도록 하는 기법: 폴링(polling), 롱 폴링(long polling), 웹소켓(WebSocket) 등

8.2.2. 폴링



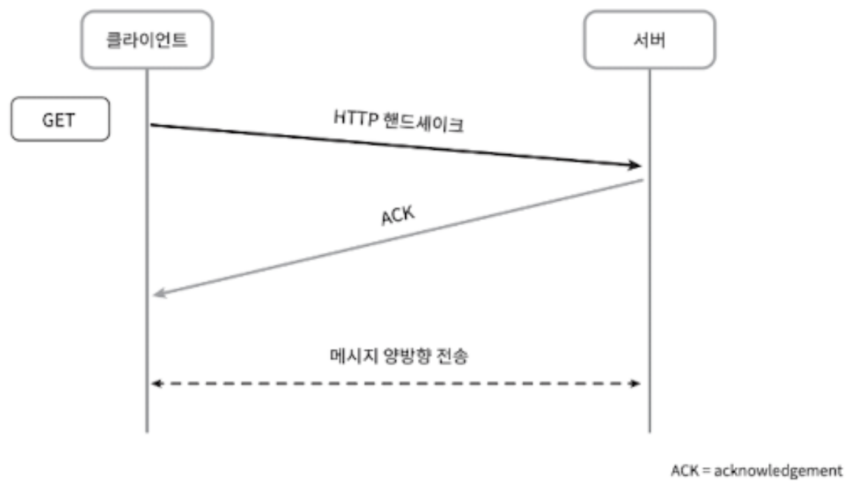
- 클라이언트가 주기적으로 서버에게 새 메시지가 있는지 물어보는 방법
- 폴링을 자주할수록 폴링 비용이 올라간다.
 - 답해줄 메시지가 없는 경우 서버 자원이 불필요하게 낭비된다.

8.2.3. 롱 폴링



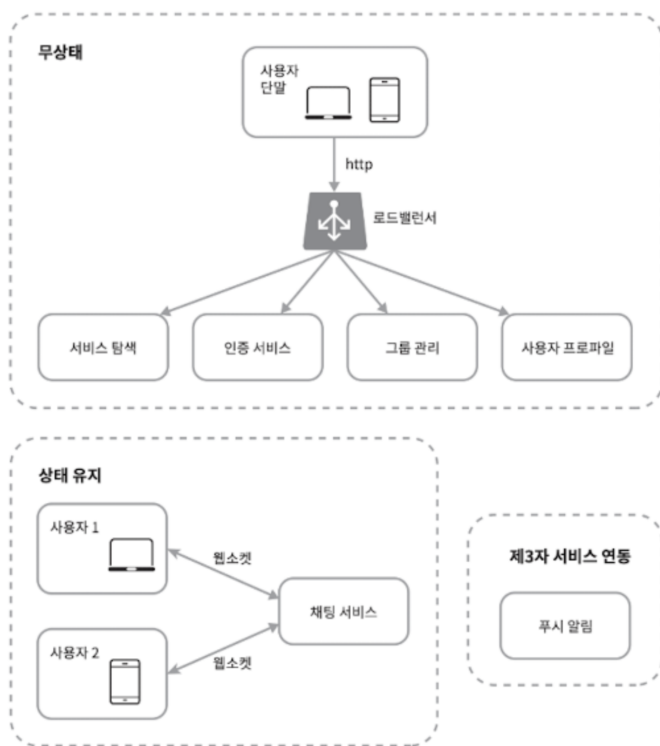
- 클라이언트는 새 메시지가 반환되거나 타임아웃 될 때까지 연결을 유지한다.
- 클라이언트가 새 메시지를 받으면 기존 연결을 종료하고 서버에 새로운 요청을 보내어 모든 절차를 다시 시작한다.
 - 메시지를 보내는 클라이언트와 수신하는 클라이언트가 같은 채팅 서버에 접속하지 않을 수도 있다.
 - HTTP 서버는 보통 무상태(stateless) 서버이다. 로드밸런싱을 위해 라운드 로빈(round robin) 알고리즘을 사용하는 경우, 메시지를 받은 서버는 해당 메시지를 수신할 클라이언트와의 롱 폴링 연결을 가지고 있지 않은 서버일 수 있다.
 - 서버 입장에서 클라이언트가 연결을 해제했는지 아닌지 알 좋은 방법이 없다.
 - 비효율적이다. 메시지를 많이 받지 않는 클라이언트도 타임아웃이 일어날 때마다 주기적으로 서버에 다시 접속한다.

8.2.4. 웹소켓

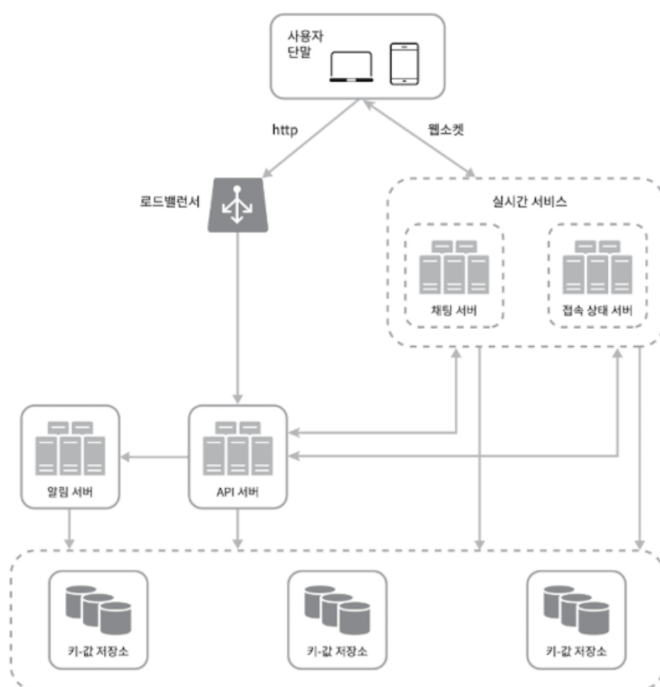


- 서버가 클라이언트에게 비동기(async) 메시지를 보낼 때 가장 널리 사용한다.
- 클라이언트가 웹소켓 연결을 시작한다.
 - 한번 맺어진 연결은 항구적이고 양방향이다.
 - 처음에는 HTTP 연결이지만 특정 핸드셰이크 절차를 거쳐 웹소켓 연결로 업그레이드된다.
 - 80, 422처럼 HTTP 또는 HTTPS 프로토콜이 사용하는 기본 포트번호를 그대로 사용하므로 웹소켓은 일반적으로 방화벽이 있는 환경에서도 잘 동작한다.

8.2.5. 개략적 설계안



- **무상태 서비스**
 - 로그인, 회원가입, 사용자 프로필 표시 등을 처리하는 전통적인 요청/응답 서비스
 - 로드밸런서 뒤에 위치한다.
 - 로드밸런서가 하는 일은 요청을 그 경로에 맞는 서비스로 정확하게 전달하는 것이다.
- **상태 유지 서비스**
 - 채팅 서비스에서 유일하게 상태 유지가 필요하다.
 - 각 클라이언트가 채팅 서버와 독립적인 네트워크 연결을 유지해야 한다.
 - 클라이언트는 보통 서버가 살아있는 한 다른 서버로 연결을 변경하지 않는다.
 - 서비스 탐색 서비스는 채팅 서비스와 협력하여 특정 서버에 부하가 몰리지 않도록 한다.
- **제 3자 서비스 연동**
 - 채팅 앱에서 가장 중요한 제3자 서비스는 푸시 알림이다.
 - 새 메시지를 받으면 앱이 실행 중이지 않더라도 알림을 받아야 한다.
- **규모 확장성**



- 동시 접속자가 1M이고, 접속당 10K의 서버 메모리가 필요하다고 가정할 때, 10GB 메모리만 있으면 모든 연결을 다 처리할 수 있을 것이다.
- 하지만 SPOF(Single-Point-Of-Failure) 때문에 이정도 규모의 트래픽을 서버 한 대로 처리하지 않는다.
 - 서버 한대만 갖는 설계안에서 출발하여 점차 다듬어 나간다.
- 실시간으로 메시지를 주고받기 위해 클라이언트는 채팅 서버와 웹소켓 연결을 끊지 않고 유지해야 한다.
 - 채팅 서버는 클라이언트 사이에 메시지를 중계하는 역할을 담당한다.
 - 접속상태 서버(presence server)는 사용자의 접속 여부를 관리한다.
 - API 서버는 로그인, 회원가입, 프로필 변경 등 그 외 나머지 전부를 처리한다.
 - 알림 서버는 푸시 알림을 보낸다.
 - 키-값 저장소(key-value store)에는 채팅 이력(chat history)을 보관한다. 시스템에 접속한 사용자는 이전 채팅 이력을 전부 보게 된다.

• 저장소

- 어떤 데이터베이스를 쓰느냐가 중요하다.
 - 데이터의 유형과 읽기/쓰기 연산의 패턴을 따져봐야 한다.
- 채팅 시스템이 다루는 데이터는 보통 두 가지이다.
 - 사용자 프로필, 설정, 친구 목록처럼 일반적인 데이터
 - 안정성을 보장하는 관계형 데이터베이스에 보관한다. 관계형 데이터베이스는 다중화(replication)와 샤딩(sharding)은 이런 데이터의 가용성과 규모 확장성을 보증하기 위해 보편적으로 사용된다.
 - 채팅 이력(chat history)
 - 읽기/쓰기 연산 패턴
 - 채팅 이력 데이터의 양은 엄청나다. 페이스북 메신저나 왓츠앱은 매일 600억(60 billion) 개의 메시지를 처리한다.
 - 빈번하게 사용되는 데이터는 주로 최근에 주고받은 메시지이다. 대부분의 사용자는 오래된 메시지를 들여다보지 않는다.
 - 검색 기능을 이용하거나, 특정 사용자가 언급(mention)된 메시지를 보거나, 특정 메시지로 점프(jump)하거나 하여 무작위적인 데이터 접근을 하게 되는 경우도 지원해야 한다.
 - 1:1 채팅 앱을 경우 읽기:쓰기 비율은 대략 1:1 정도이다.
- 키-값 저장소를 채택한다.
 - 키-값 저장소는 수평적 규모확장(horizontal scaling)이 쉽다.
 - 키-값 저장소는 데이터 접근 지연시간(latency)이 낮다.
 - 관계형 데이터베이스는 데이터 가운데 롱 테일(long tail)에 해당하는 부분을 잘 처리하지 못하는 경향이 있다.
 - 인덱스가 커지면 데이터에 대한 무작위적 접근(random access)을 처리하는 비용이 늘어난다.

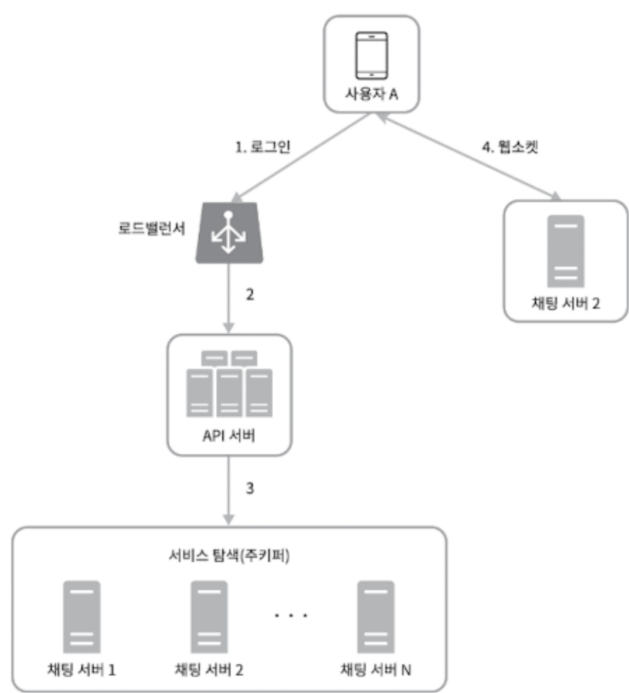
- 많은 채팅 시스템이 키-값 저장소를 채택하고 있다. 페이스북 메시저는 HBase를 사용하고 디스코드는 카산드라(Cassandra)를 사용한다.

8.2.6. 데이터 모델

- 1:1 채팅을 위한 메시지 테이블
 - 기본 키(primary key): `message_id`
 - 메시지 순서를 쉽게 정할 수 있도록 하는 역할도 담당한다.
 - `created_at`은 동시에 만들어질 수 있으므로 메시지 순서를 정할 수 없다.
- 그룹 채팅을 위한 메시지 테이블
 - 기본 키(primary key): `(channel_id, message_id)`의 복합 키(composite key)
 - `channel`: 채팅 그룹
 - `channel_id`는 파티션 키(partition key)로도 사용한다. 그룹 채팅에 적용될 모든 질의는 특정 채널을 대상으로 한다.
- 메시지 ID
 - `message_id` 값은 고유해야 한다.(uniqueness)
 - ID 값은 정렬 가능해야 하며 시간 순서와 일치해야 한다. 새로운 ID는 이전 ID보다 큰 값이어야 한다.
 - 위의 두 조건을 만족하는 방법
 - RDBMS는 `auto_increment`가 대안이 될 수 있다. 하지만 NoSQL은 해당 기능을 제공하지 않는다.
 - 스노플레이크 같은 전역적 64-bit 순서 번호(sequence number) 생성기를 이용한다.
 - 지역적 순서 번호 생성기(local sequence number generator)를 이용한다.

8.3. 3단계: 상세 설계

8.3.1. 서비스 탐색

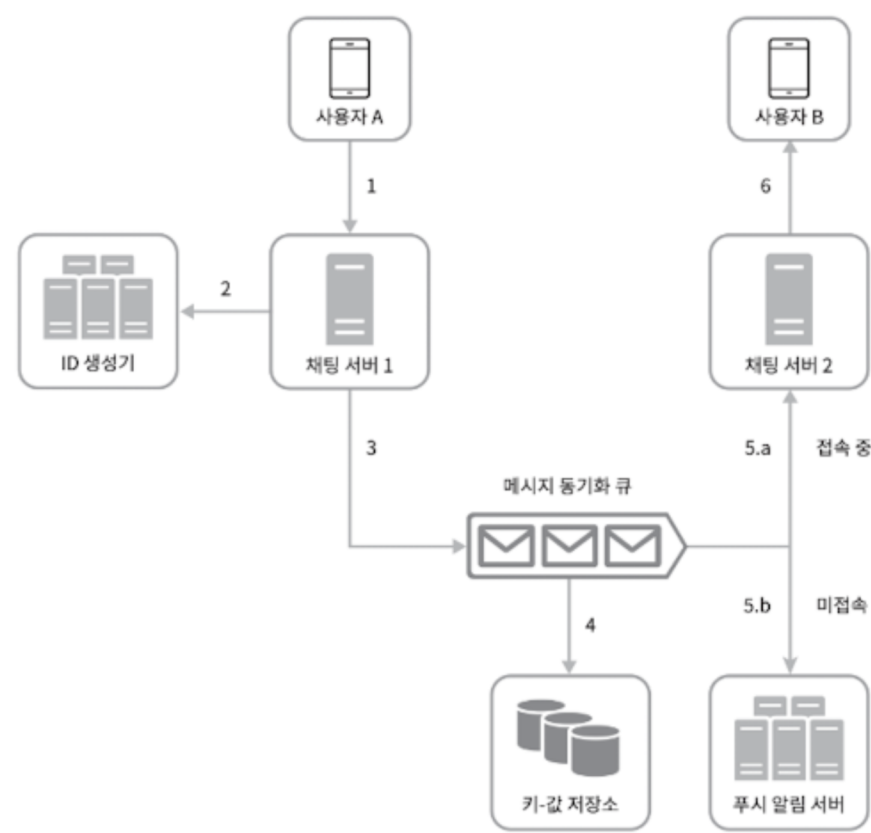


- 서비스 탐색 기능의 주된 역할은 클라이언트에게 가장 적합한 채팅 서버를 추천하는 것이다.
 - 사용되는 기준: 클라이언트의 위치(geographical location), 서버의 용량(capacity) emd
 - 서비스 탐색 기능을 구현하는 데 널리 쓰이는 오픈 소스 솔루션: 아파치 주키퍼(Apache Zookeeper)
 - 사용 가능한 모든 채팅 서버를 여기 등록시켜 두고, 클라이언트가 접속을 시도하면 사전에 정한 기준에 따라 최적의 채팅 서버를 골라준다.
1. 사용자 A가 시스템에 로그인을 시도한다.
 2. 로드밸런서가 로그인 요청을 API 서버들 가운데 하나로 보낸다.
 3. API 서버가 사용자 인증을 처리하고 나면 서비스 탐색 기능이 동작하여 해당 사용자를 서비스할 최적의 채팅 서버를 찾는다. 여기서 채팅 서버 2가 선택되어 사용자 A에게 반환되었다고 가정해보자.

4. 사용자 A는 채팅 서버 2와 웹소켓 연결을 맺는다.

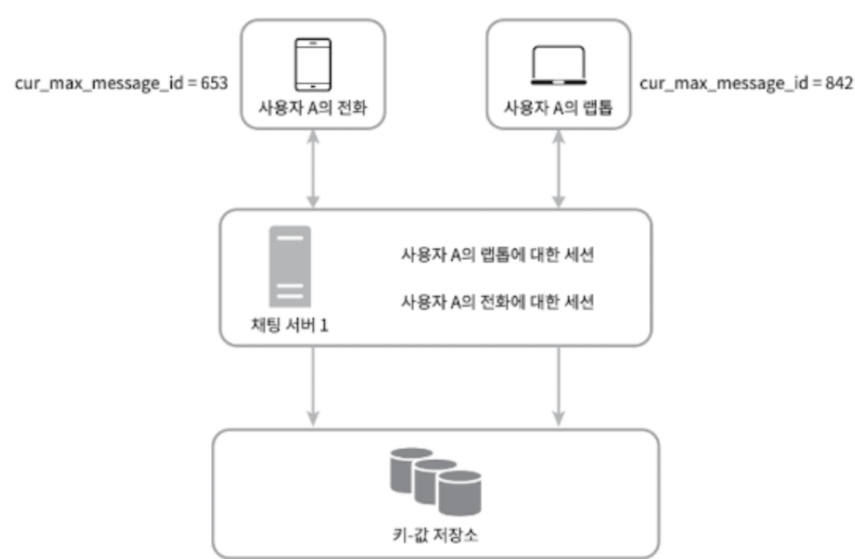
8.3.2. 메시지 흐름

• 1:1 채팅 메시지 처리 흐름



- 1. 사용자 A가 채팅 서버 1로 메시지를 전송한다.
- 2. 채팅 서버 1은 ID 생성기를 사용해 해당 메시지의 ID를 결정한다.
- 3. 채팅 서버 1은 해당 메시지를 메시지 동기화 큐로 전송한다.
- 4. 메시지가 키-값 저장소에 보관된다.
- 5. 접속 중/미접속 중
 - a. 사용자 B가 접속 중인 경우 메시지는 사용자 B가 접속 중인 채팅 서버로 전송된다.
 - b. 사용자 B가 접속 중이 아닌 경우 푸시 알림 메시지를 푸시 알림 서버로 보낸다.
- 6. 채팅 서버 2는 메시지를 사용자 B에게 전송한다. 사용자 B와 채팅 서버 2 사이에는 웹소켓 연결이 있는 상태이므로 그것을 사용한다.

• 여러 단말 사이의 메시지 동기화

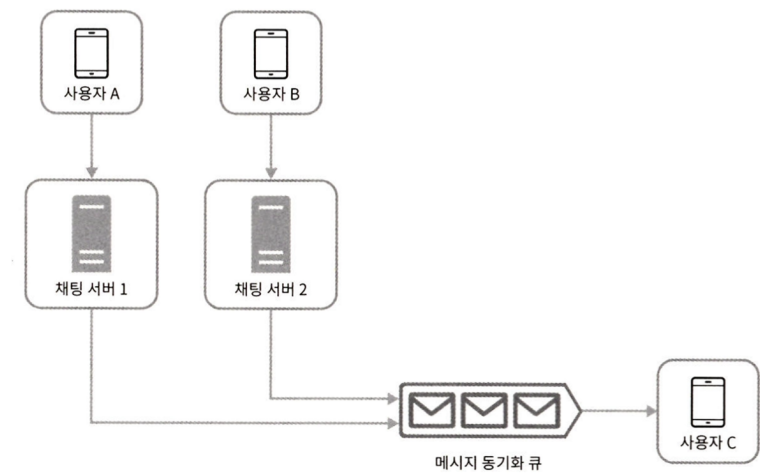


- 전화기와 랩톱 두 대의 단말을 이용한다.
 - 사용자 A가 전화기에서 채팅 앱에 로그인한 결과로 채팅 서버 1과 해당 단말 사이에 웹소켓 연결이 만들어져 있고, 랩톱에서 로그인한 결과로 별도 웹소켓이 채팅 서버 1에 연결되어 있다.
 - 각 단말은 `cur_max_message_id` 변수를 유지한다.
 - 해당 단말에서 관측된 가장 최신 메시지의 ID를 추적한다.
 - 새 메시지로 간주하는 조건
 - 수신자 ID가 현재 로그인한 사용자 ID와 같다.

- 키-값 저장소에 보관된 메시지로써, ID가 `cur_max_message_id` 보다 크다.
- 소규모 그룹 채팅에서의 메시지 흐름

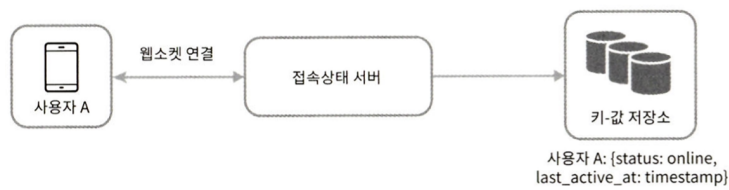


- 사용자 A가 채팅 방에서 메시지를 보냈을 때 발생하는 일
 - 사용자 A가 보낸 메시지가 사용자 B와 C의 메시지 동기화 큐(message sync queue)에 복사된다.
 - 이 설계안은 소규모 그룹 채팅에 적합하다.
 - 새로운 메시지가 왔는지 확인하려면 자기 큐만 보면 되므로 메시지 동기화 플로가 단순하다.
 - 그룹이 크지 않으면 메시지를 수신자별로 복사해서 큐에 넣는 작업의 비용이 문제가 되지 않는다.
 - 많은 사용자를 지원해야 하는 경우 똑같은 메시지를 모든 사용자의 큐에 복사하는게 바람직하지 않다.
- 그룹 채팅에서 메시지를 수신할 때는 모든 사용자의 메시지를 받을 수 있어야 한다.



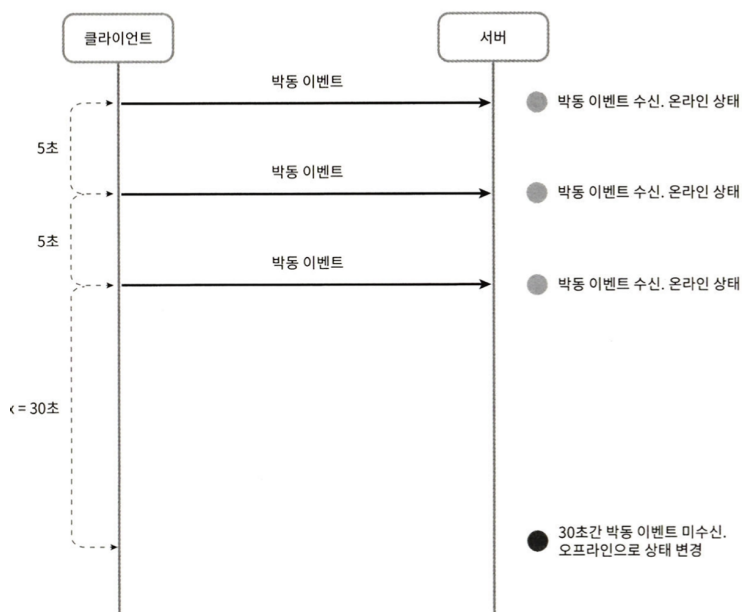
8.3.3. 접속상태 표시

- 사용자 로그인

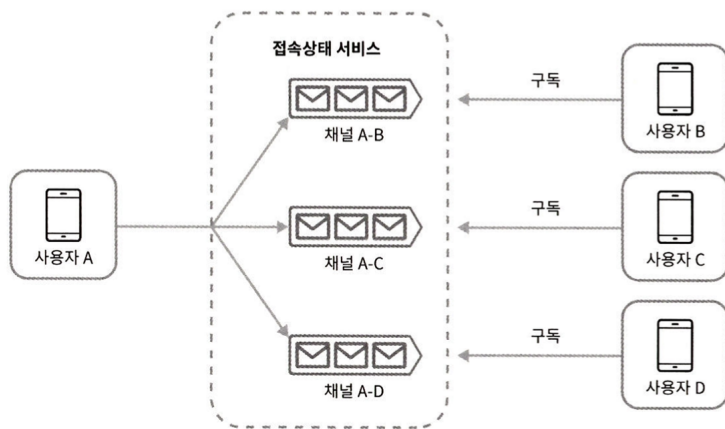


사용자와 실시간 서비스 사이에 연결이 이루어지면 접속상태 서버는 사용자의 상태와 `last_active_at` 타임스탬프를 키-값 저장소에 보관한다.

- 로그아웃
- 접속 장애



- 상태 정보의 전송



발행-구독 모델을 사용해 각 친구마다 채널을 하나씩 두어 상태 정보 변화를 통지받을 수 있도록 설계하면 쉽게 상태 정보 변화를 전달 받을 수 있게 된다.

- 소규모 그룹 채팅에 적합하다.
 - 대규모 그룹 채팅에서 상태 정보를 전달 기능이 필요하다면 입장 순간에만 상태 정보를 읽어가던가, 수동으로 갱신하도록 유도해야 한다.

8.4. 4단계: 마무리



추가 논의 사항

- 채팅 앱을 확장하여 사진이나 비디오 등의 미디어를 지원하도록 하는 방법
 - 사진이나 비디오 등의 미디어를 지원하기 위해서는 압축 방식, 클라우드 저장소, 섬네일 생성 등을 고려해볼 수 있다.
- 종단 간 암호화: 당사자 외에 아무도 메시지 내용을 볼 수 없게 하여 메시지 전송의 보안을 강화할 수 있다.
- 캐시: 읽은 메시지에 대해 캐시를 도입하면 서버와 주고받는 데이터 양을 줄 일 수 있다.
- 로딩 속도 개선: 사용자의 데이터, 채널 등을 지역적으로 분산하는 네트워크를 구축하면 로딩 속도를 개선할 수 있다.
- 오류 처리
 - 채팅 서버 오류
 - 메시지 재전송