EJERCICIOS PRÁCTICOS

SISTEMA DE CHAT

Daniel Blanco Calviño

REQUISITOS

- **Requisitos funcionales.**
  - Mensajes individuales y grupales.
    - Pueden contener texto, imágenes o vídeos.
  - Los usuarios podrán iniciar sesión en múltiples dispositivos a la vez.
    - Aplicación móvil y web.
  - Recibos de lectura.
  - Última hora de conexión.

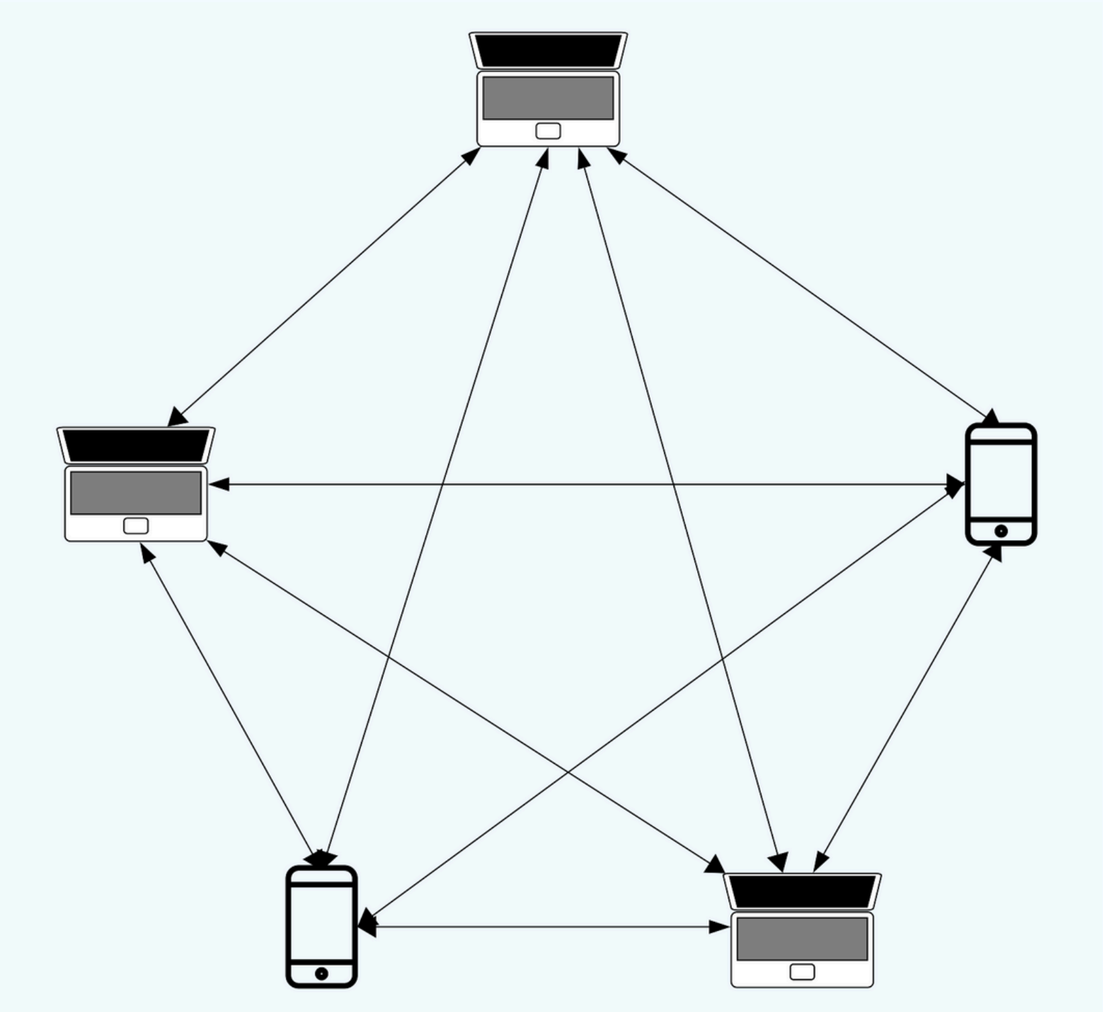
- **Requisitos no funcionales.**
  - Muy baja latencia. Lo más cercano posible al tiempo real.
  - Alta disponibilidad.

HIPÓTESIS Y ESTIMACIONES

- Límite de 10k caracteres por mensaje.
- Cada usuario envía de media 150 mensajes de 100 caracteres al día.
- Se envían 5 archivos multimedía por usuario al día, con un tamaño medio de 1 MB.
- 100 millones de usuarios activos al día.
  - 150 * 100 * 100M de bytes al día = **1.5 TB** de almacenamiento para los mensajes.
  - 1MB * 5 * 100M = **500 TB** de almacenamiento para los archivos multimedia.
  - En total nuestro sistema necesirará sobre **501.5 TB de almacenamiento al día.**

PROTOCOLO DE COMUNICACIÓN - P2P

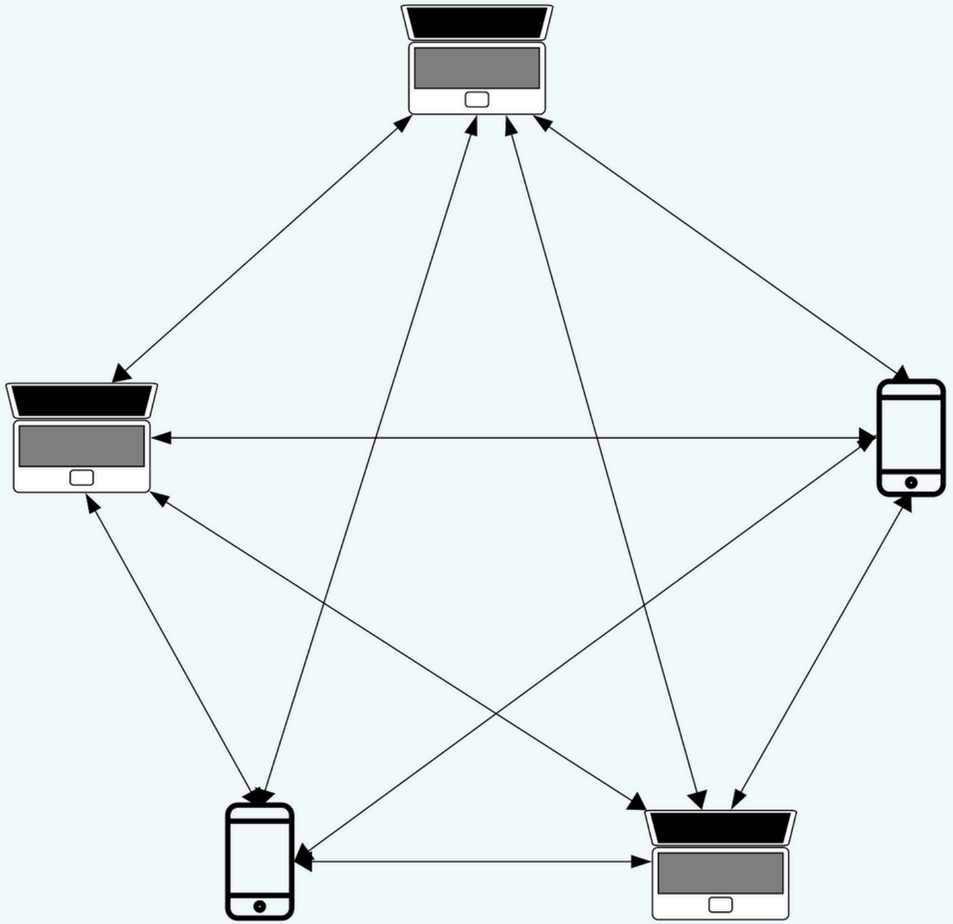
- Los clientes se comunican directamente.
  - No se necesita un servidor.



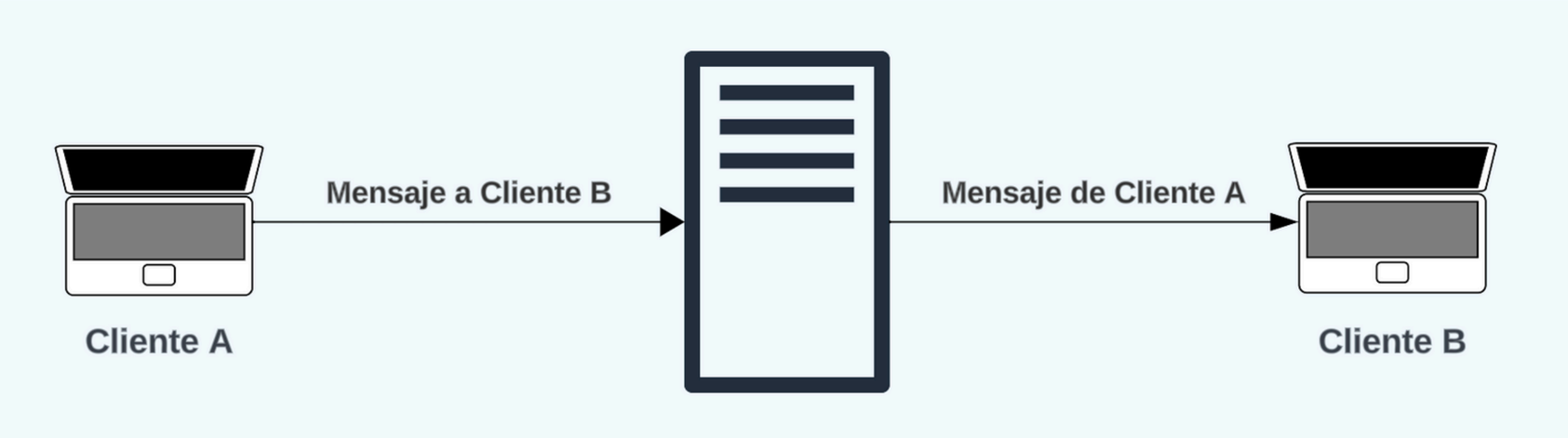
PROTOCOLO DE COMUNICACIÓN - P2P

- Los clientes se comunican directamente.
  - No se necesita un servidor.

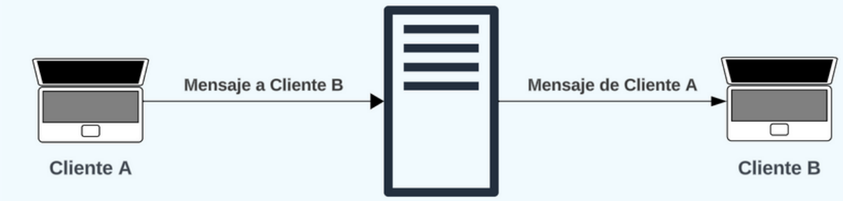
- No es el protocolo adecuado.
  - Se necesitan mantener **muchas conexiones.**
  - Las condiciones de **red** de los clientes tienden a ser más **inestables**.
  - Más complicado de asegurar el envío de mensajes a clientes sin conexión.



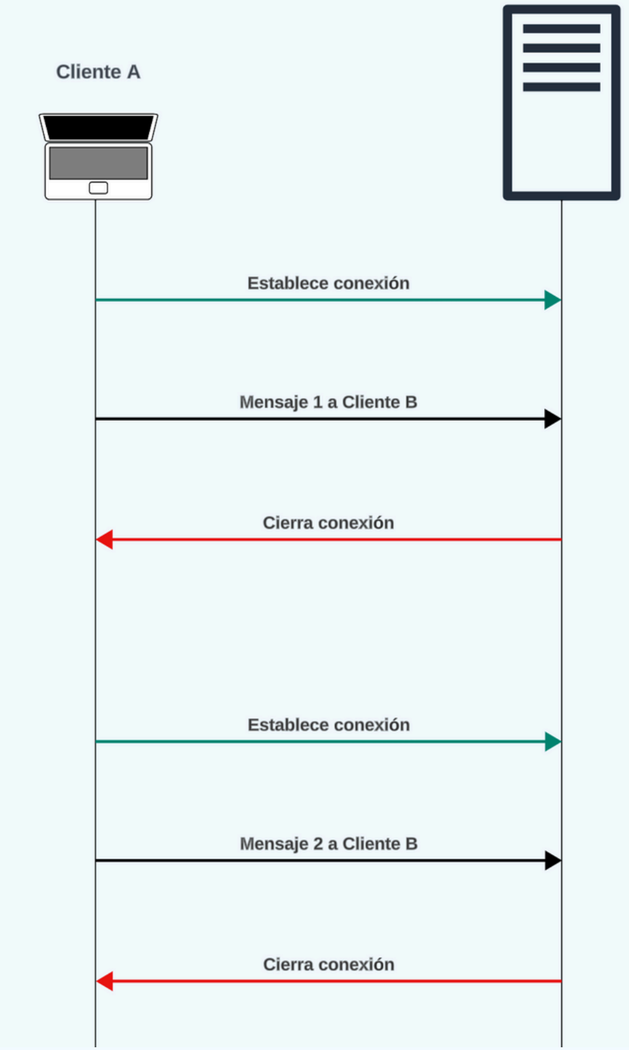
PROTOCOLO DE COMUNICACIÓN



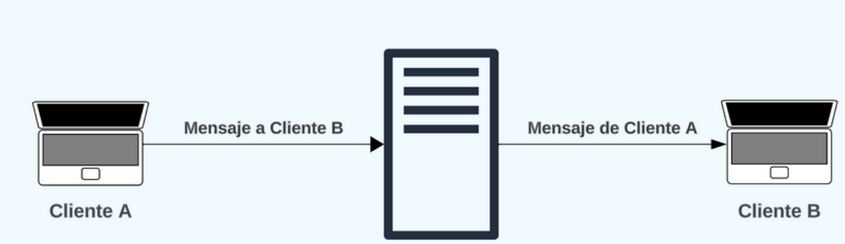
PROTOCOLO HTTP - ENVÍO DE MENSAJES



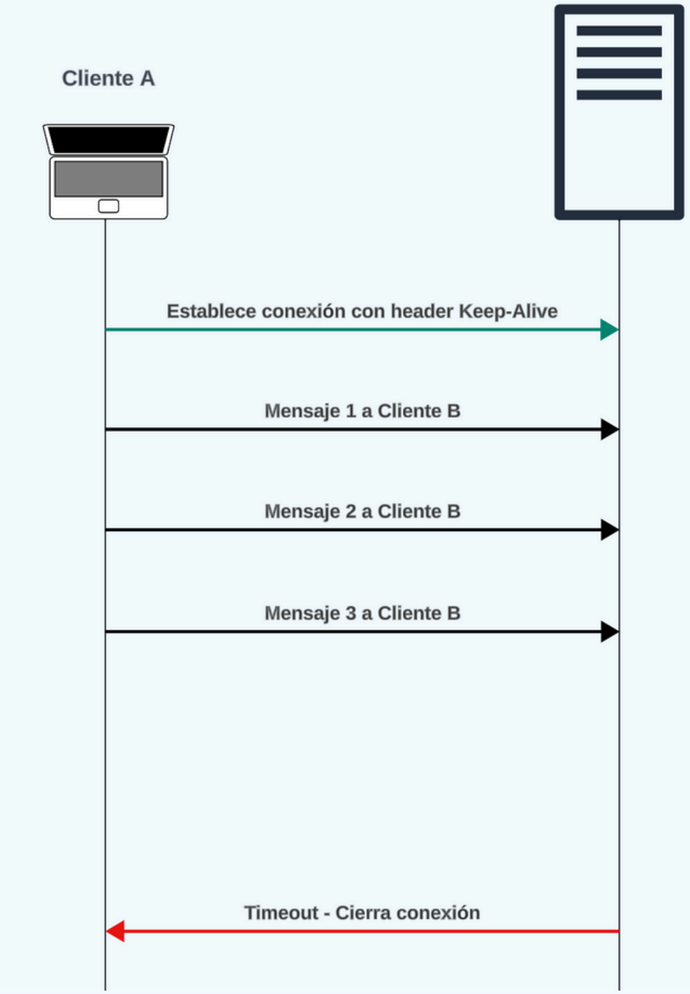
- Problema: el cliente debe establecer una nueva conexión por cada mensaje que envía.



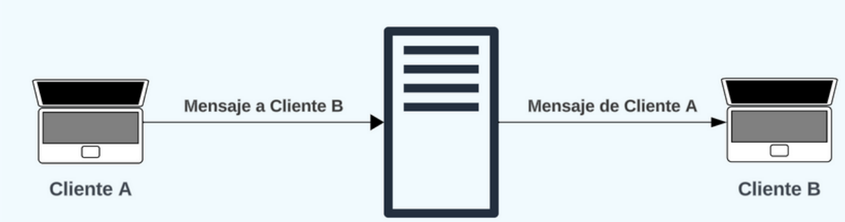
PROTOCOLO HTTP - ENVÍO DE MENSAJES



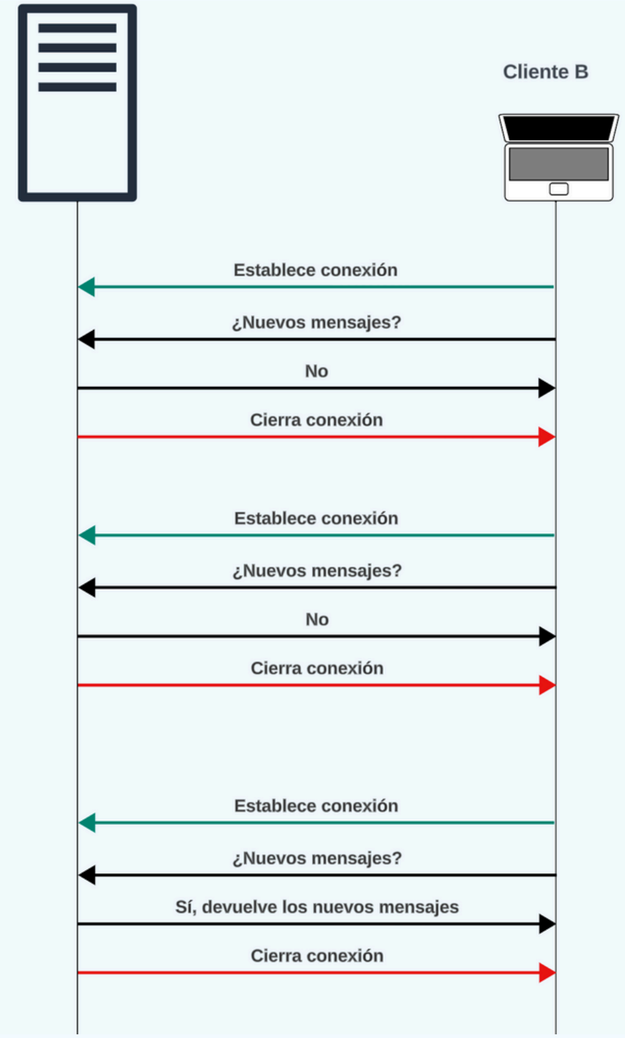
- Problema: el cliente debe establecer una nueva conexión por cada mensaje que envía.
  - Header **Keep-Alive.**



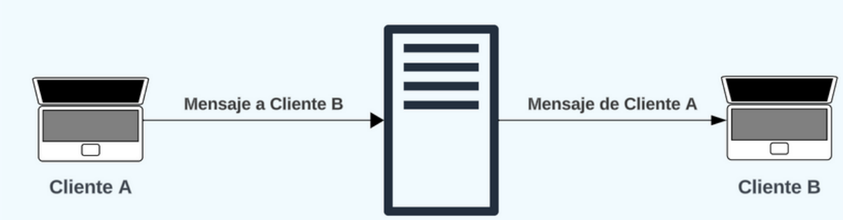
PROTOCOLO HTTP - RECEPCIÓN DE MENSAJES



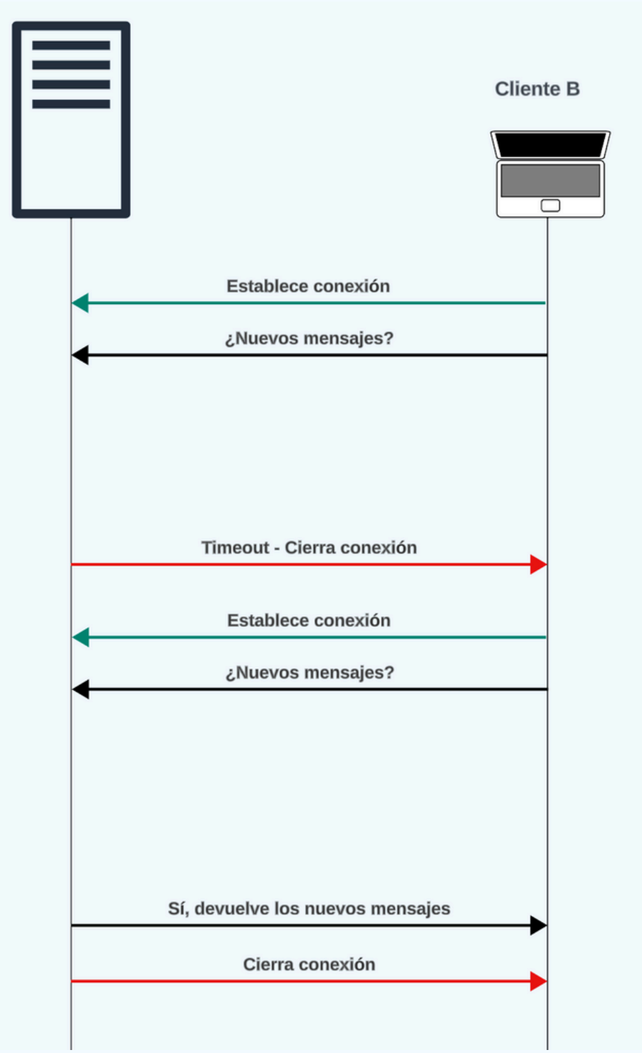
- Problema: el cliente lleva la iniciativa en HTTP.
  - Se necesita realizar **polling**.
    - **Muy poco eficiente**, se malgastan recursos.



PROTOCOLO HTTP - RECEPCIÓN DE MENSAJES

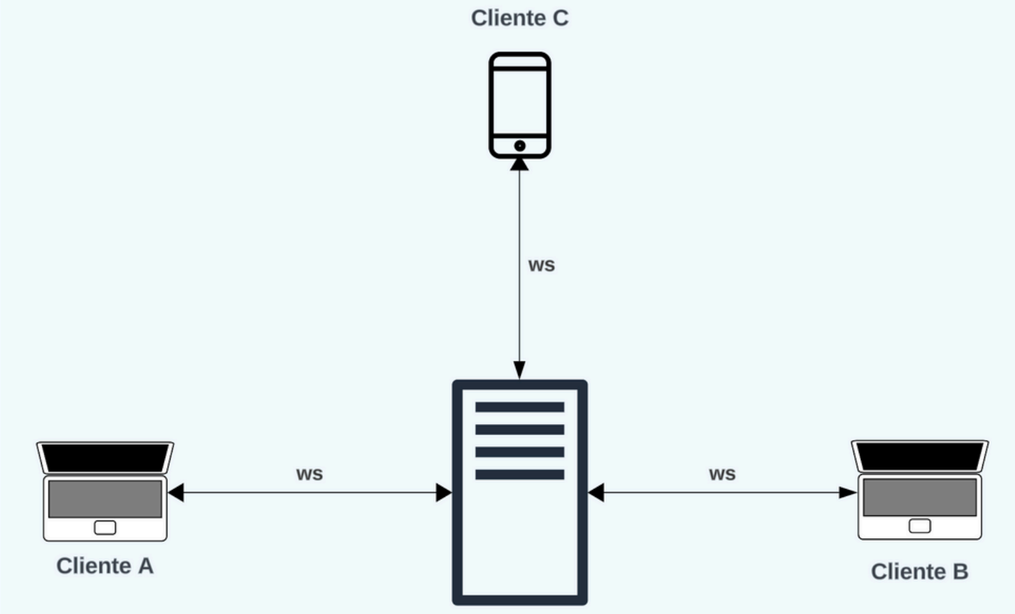


- Problema: el cliente lleva la iniciativa en HTTP.
  - Se necesita realizar **polling**.
    - **Muy poco eficiente**, se malgastan recursos.
  - Variante: **long-polling.**
    - Se mantiene la conexión abierta hasta que haya nuevos mensajes o hasta llegar a un timeout. Se ahorran conexiones.

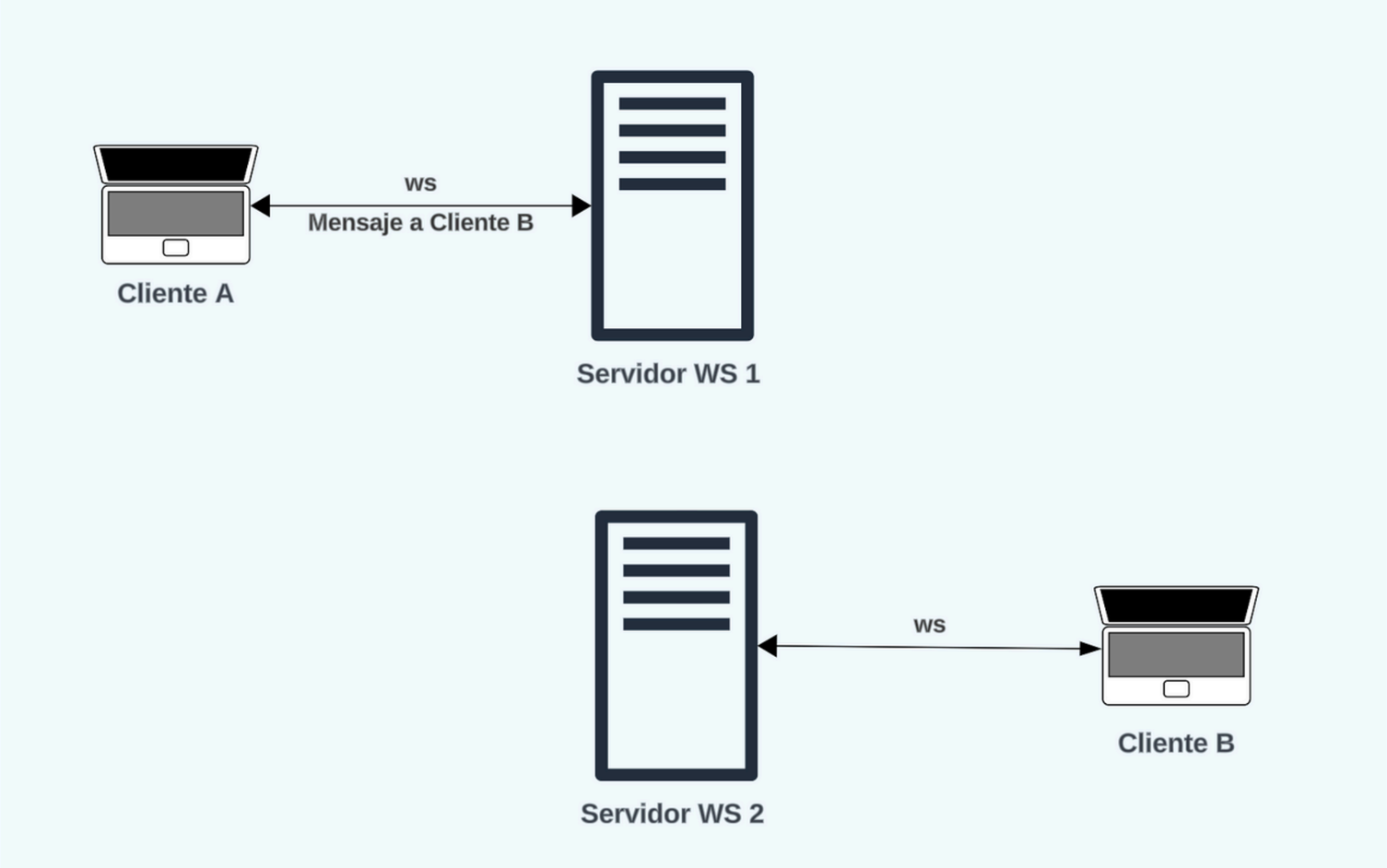


PROTOCOLO WEBSOCKET

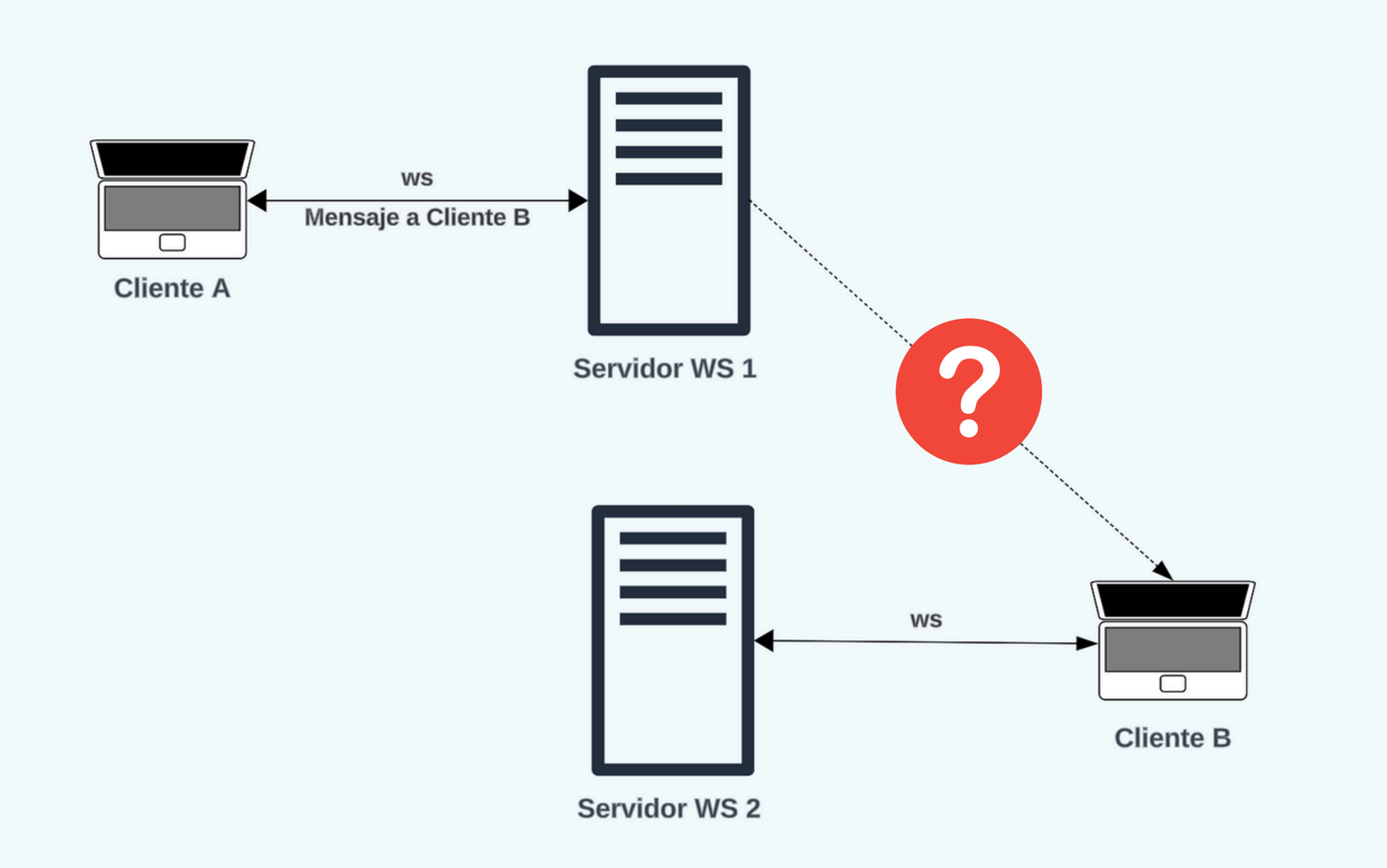
- Necesitamos **comunicación bidireccional continua.**
  - Protocolo **WebSocket.**
  - Conexión **persistente** en el tiempo.
  - Mucho más eficiente al evitar la necesidad de polling.
  - **No es stateless.** El servidor al que se conecta un cliente se mantiene fijo mientras la conexión viva.

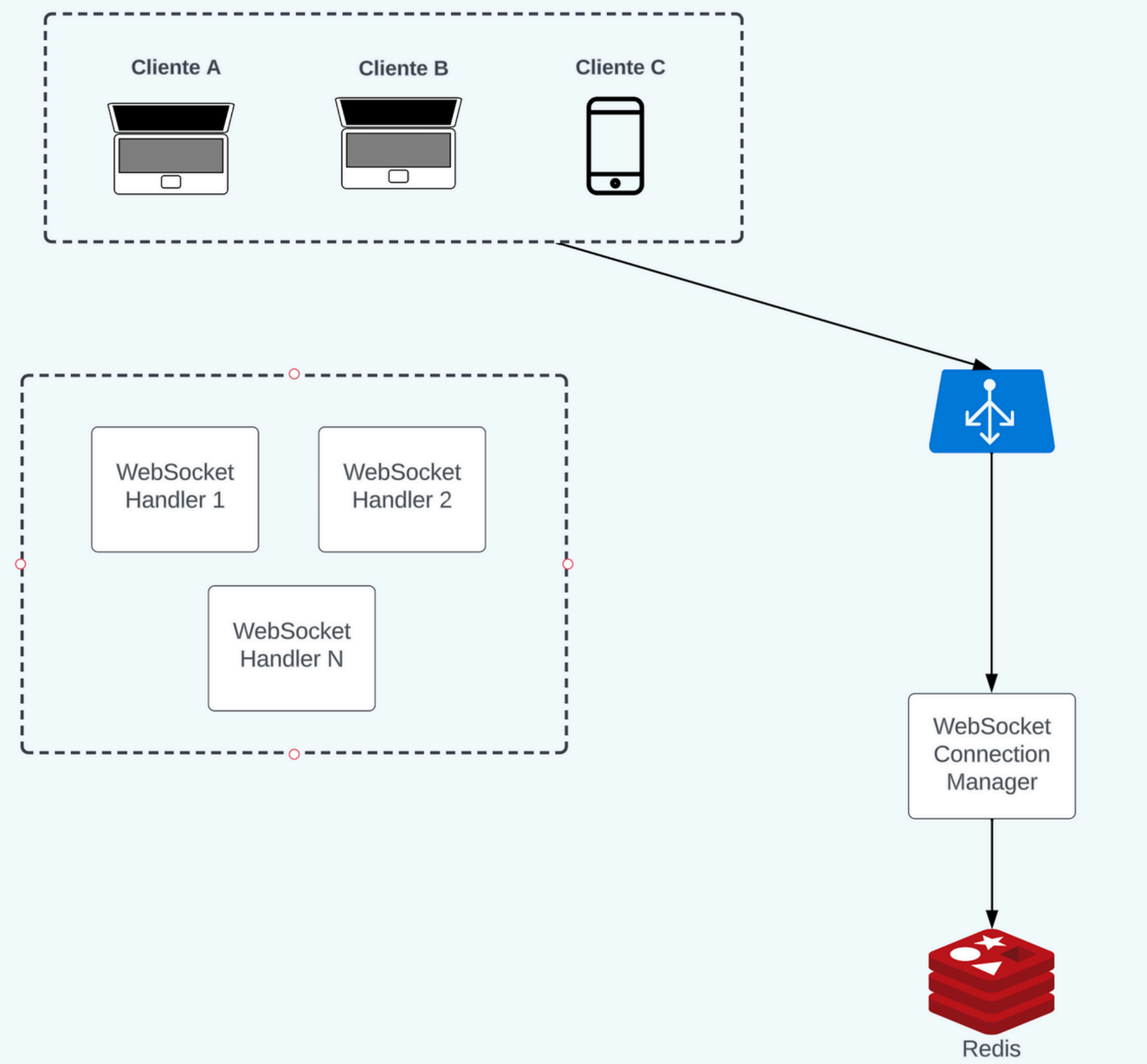


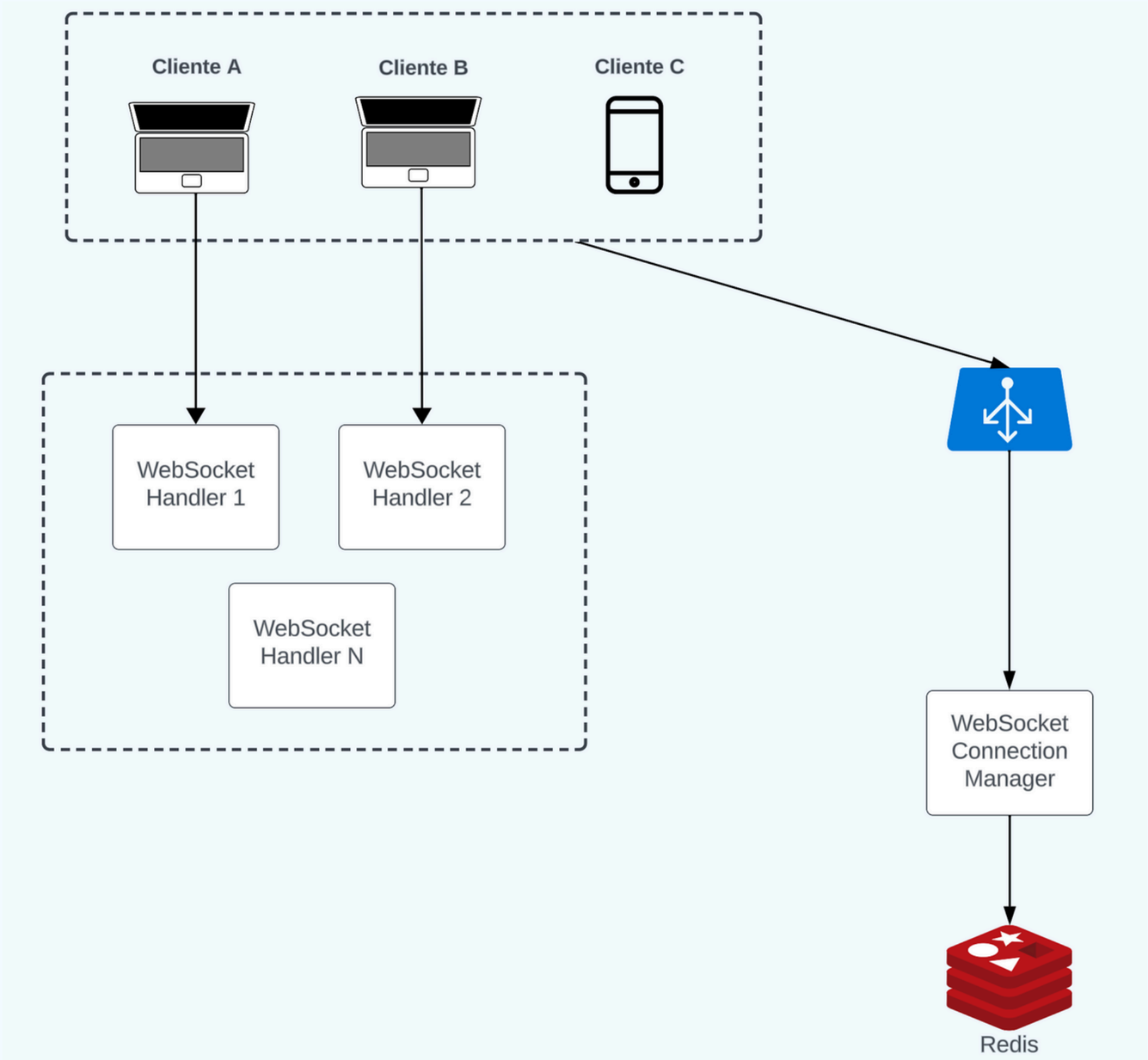
PROTOCOLO WEBSOCKET

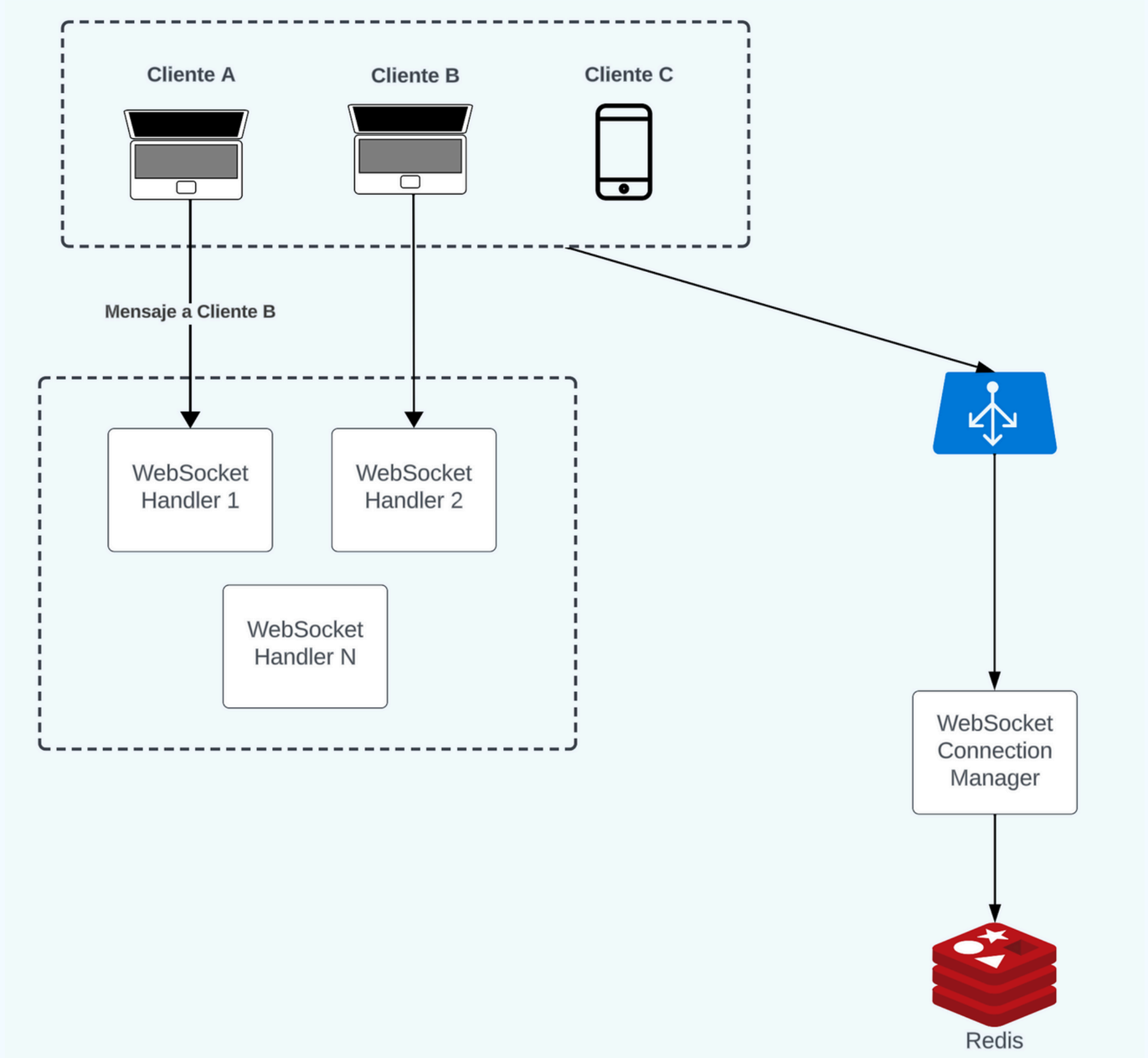


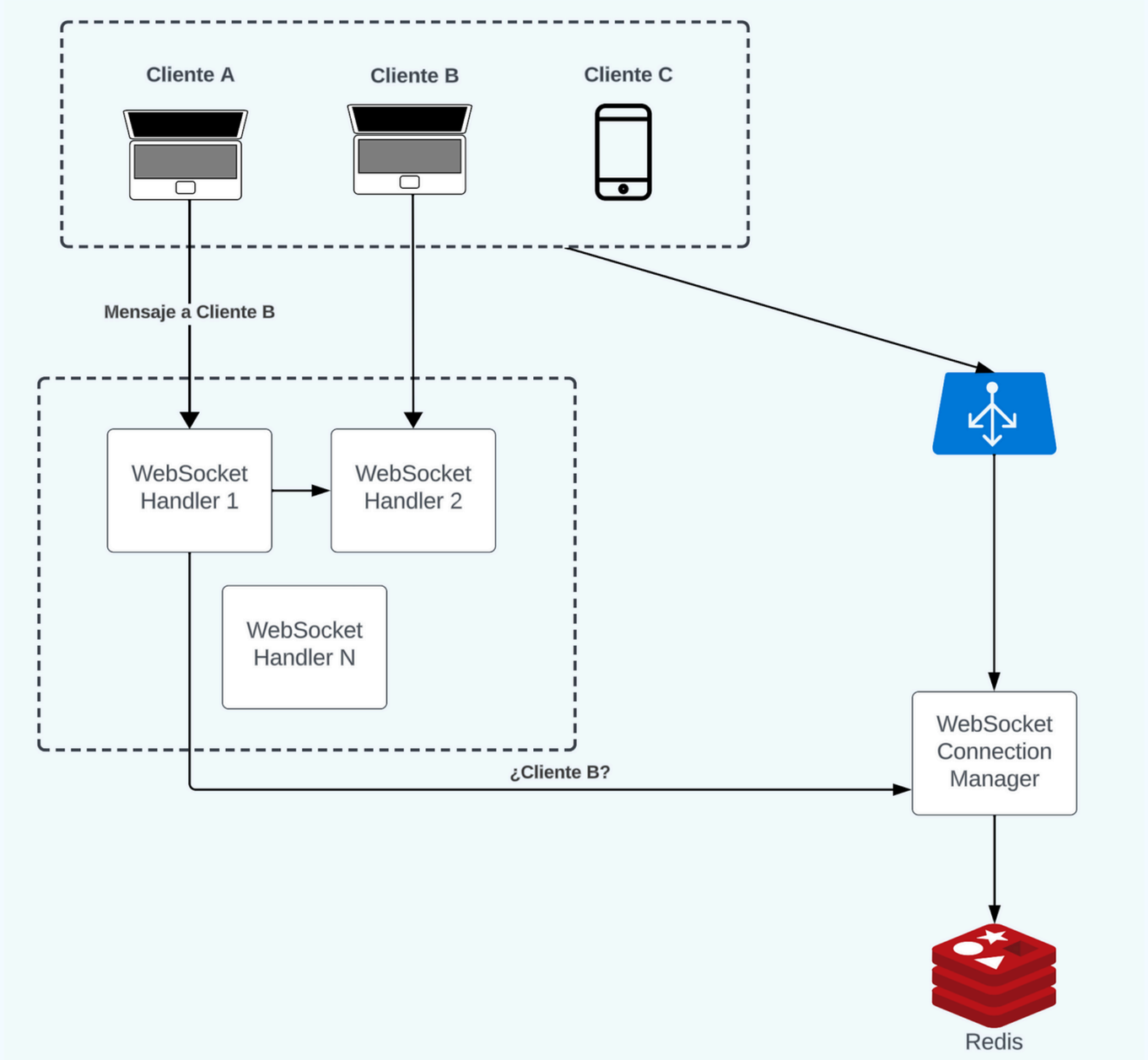
PROTOCOLO WEBSOCKET

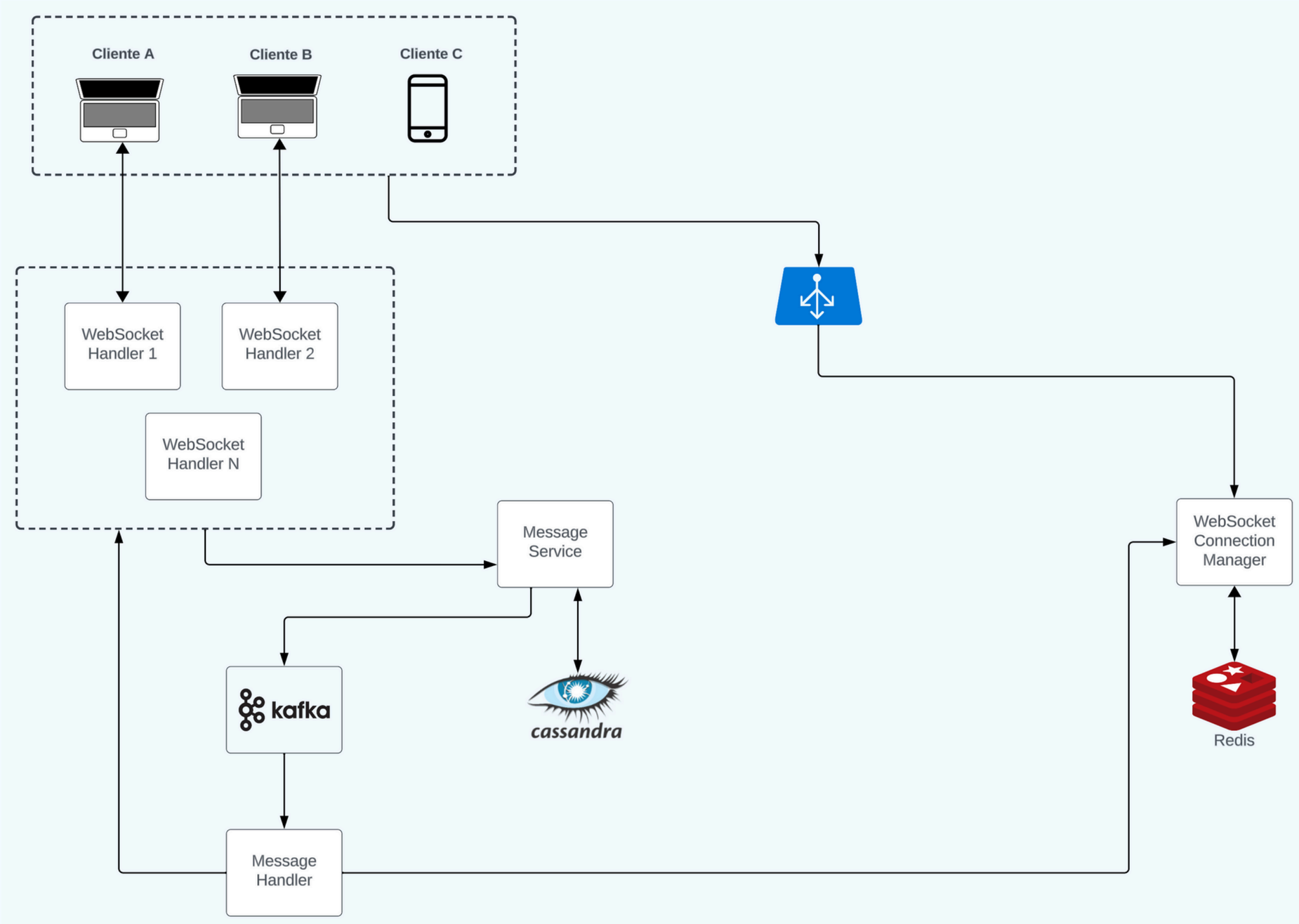


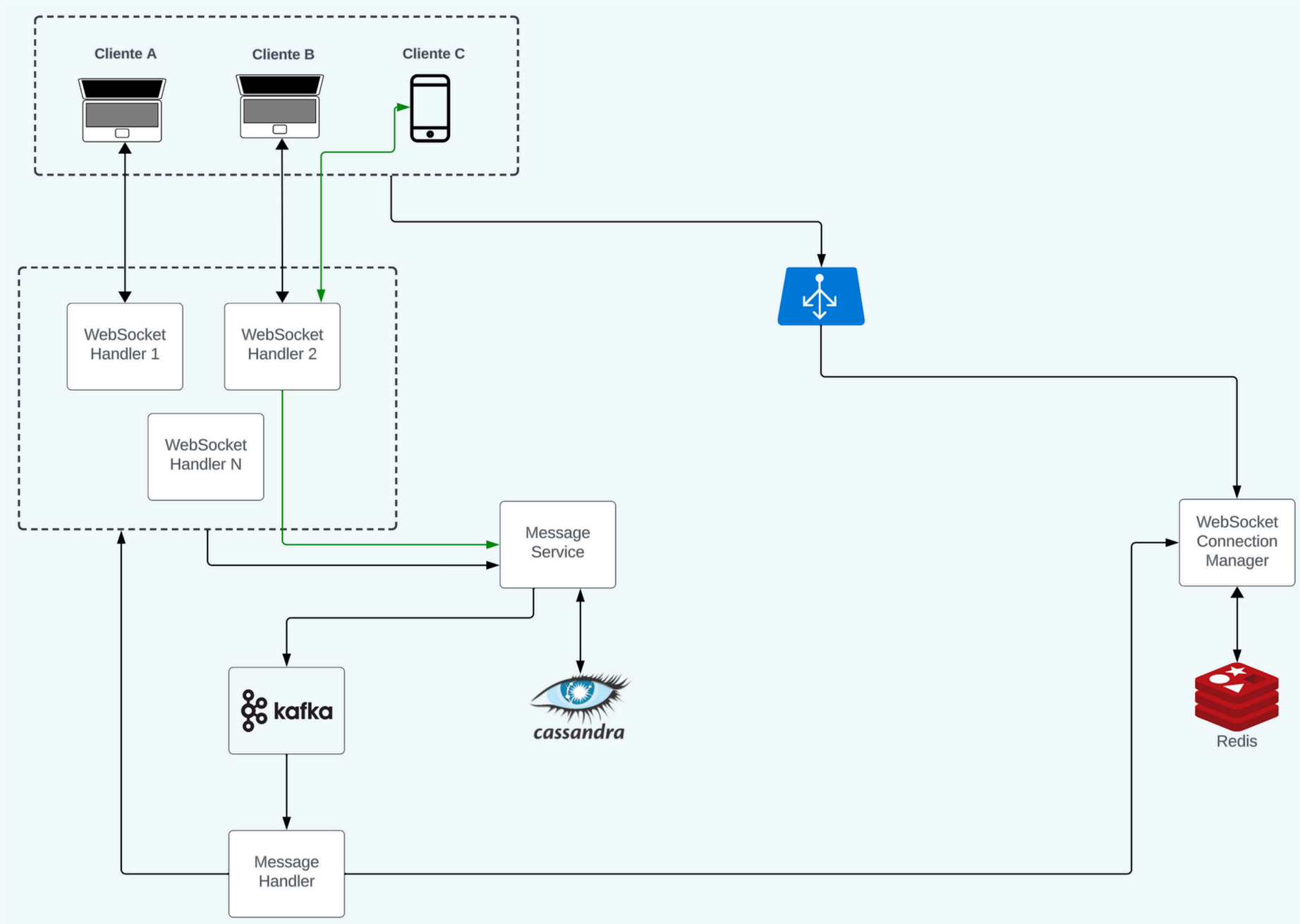

Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
WebSocket Connection Manager
Redis

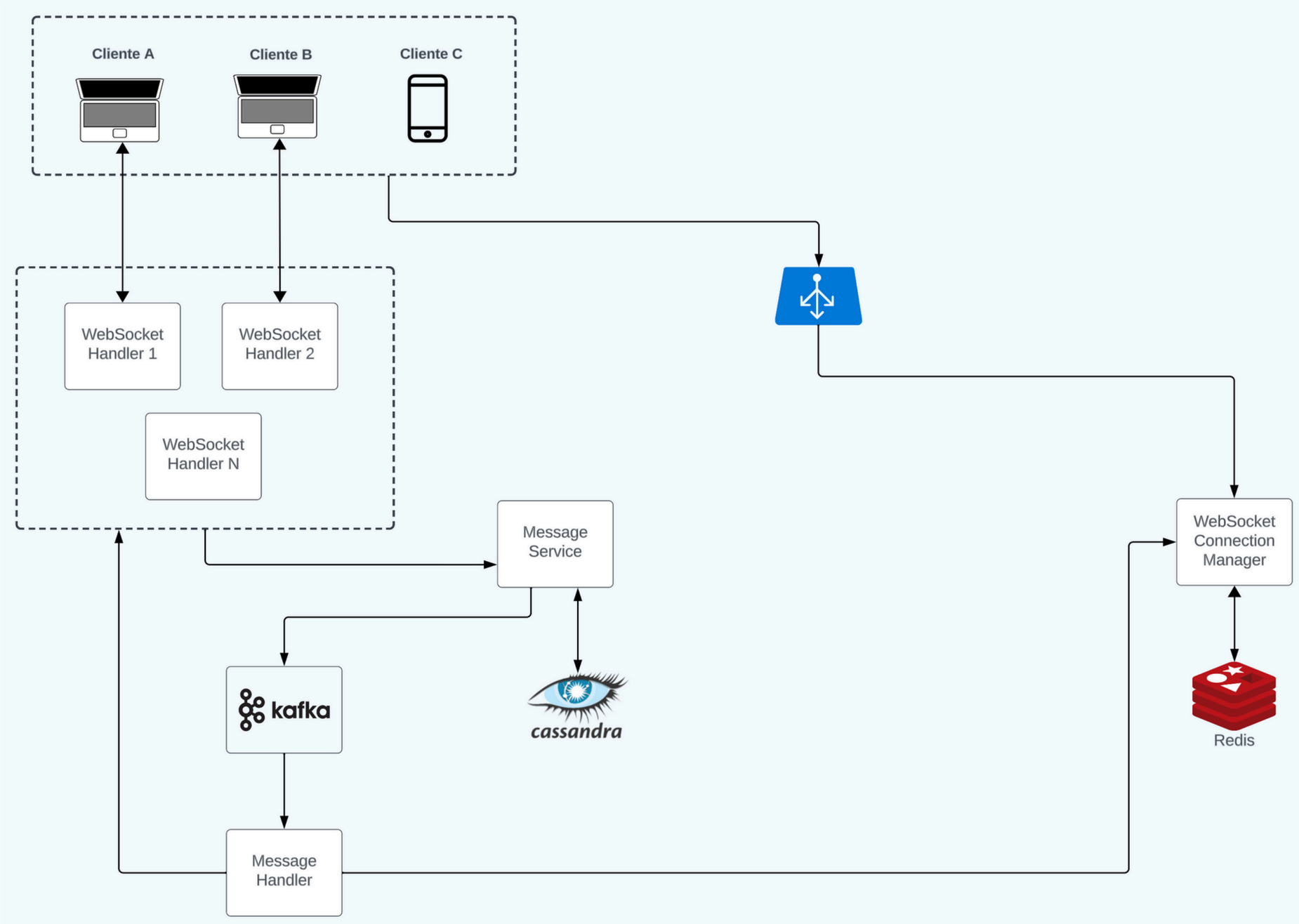

Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
WebSocket Connection Manager
Redis

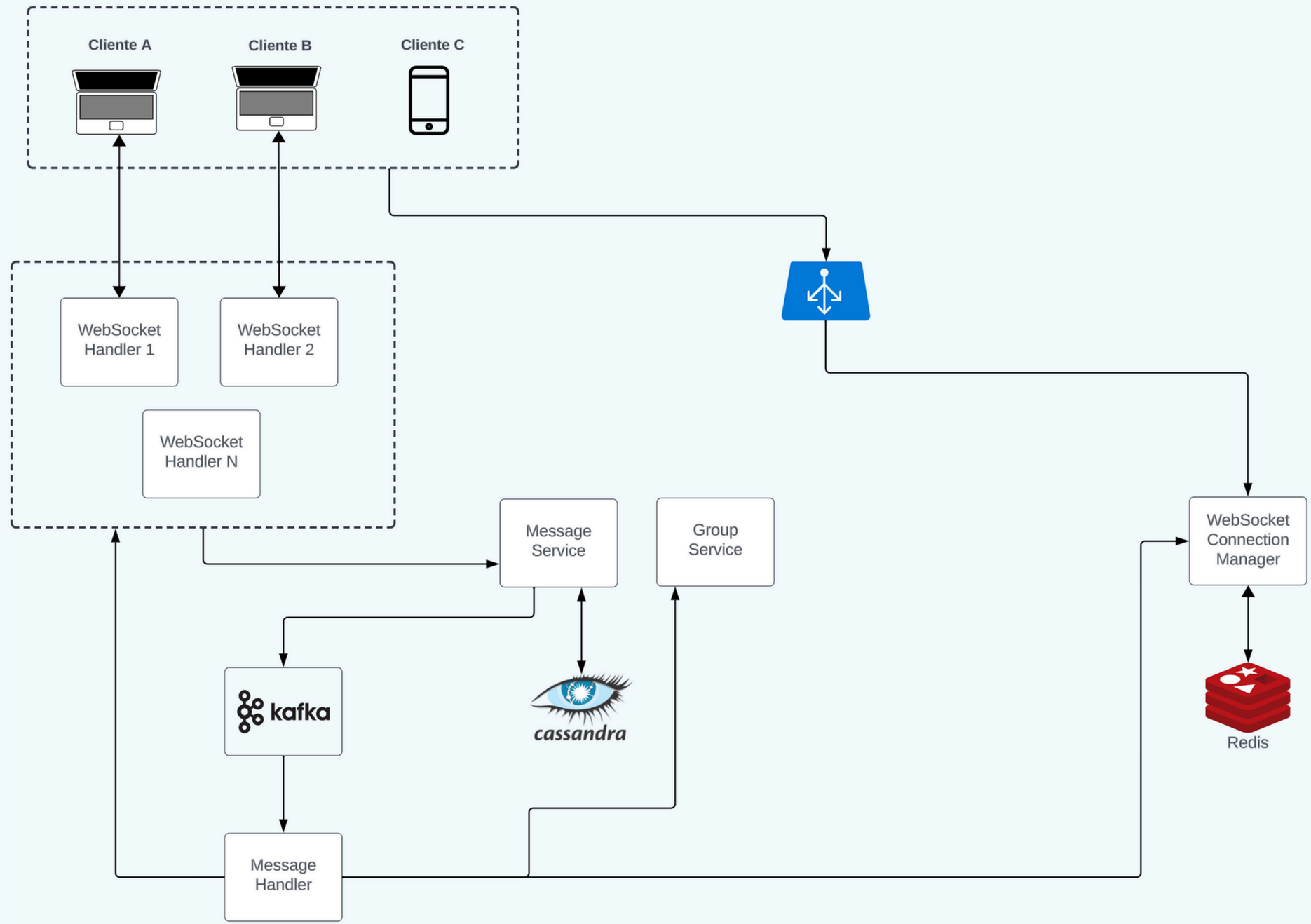

Cliente A
Cliente B
Cliente C
Mensaje a Cliente B
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
WebSocket Connection Manager
Redis

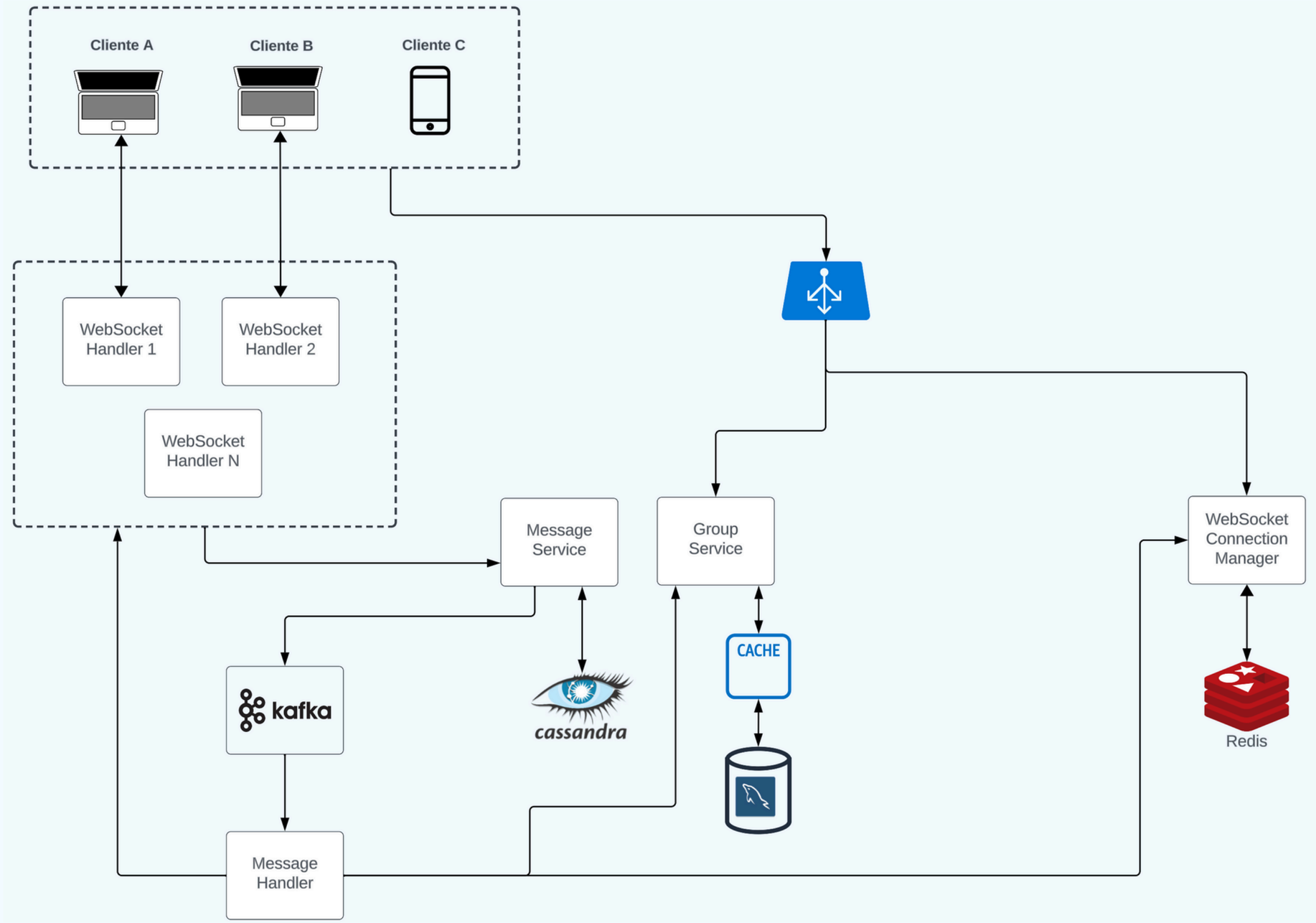

Cliente A
Cliente B
Cliente C
Mensaje a Cliente B
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
¿Cliente B?
WebSocket Connection Manager
Redis

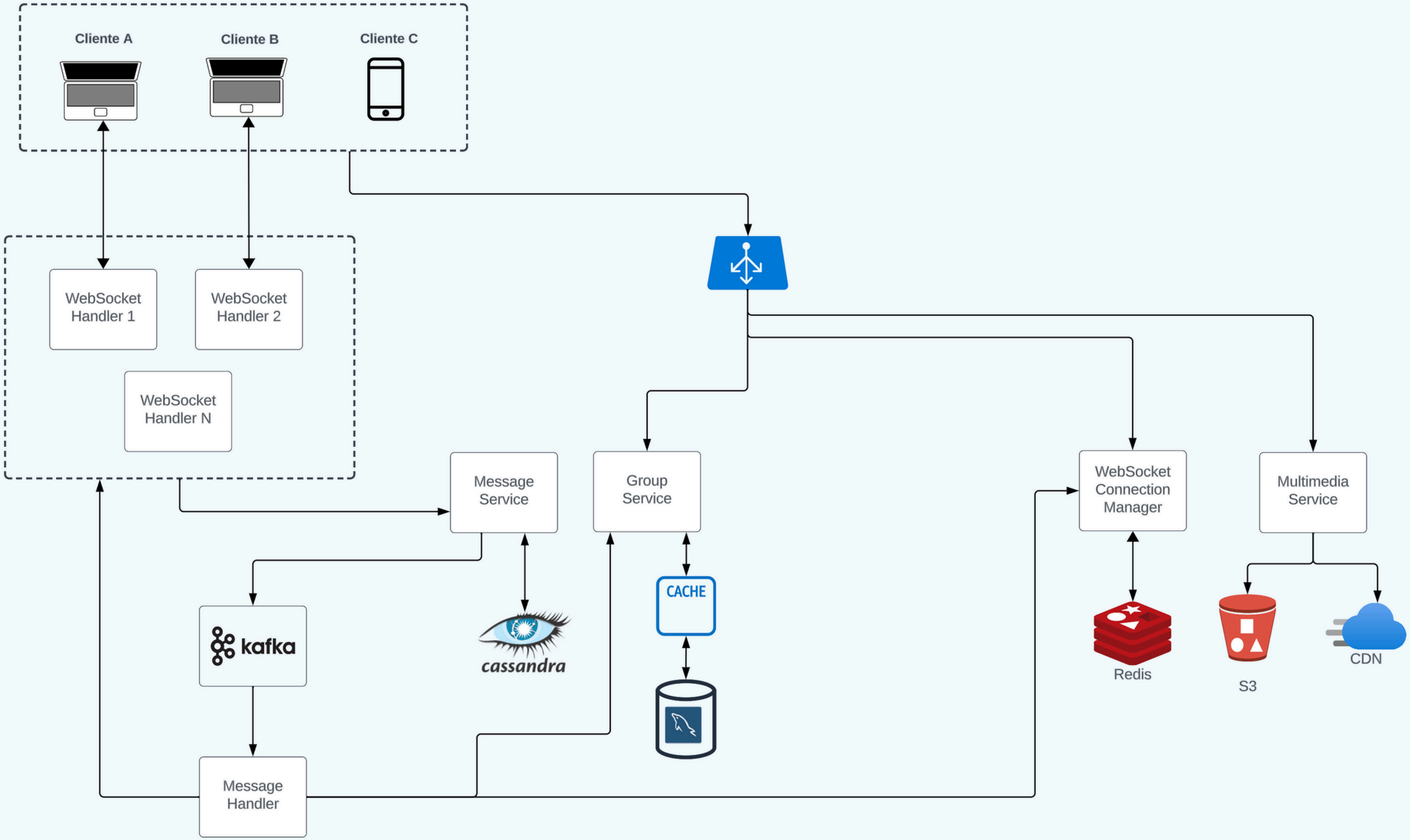

Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
Message Service
WebSocket Connection Manager
kafka
cassandra
Redis
Message Handler


Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
Message Service
WebSocket Connection Manager
kafka
cassandra
Redis
Message Handler


Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
Message Service
WebSocket Connection Manager
kafka
cassandra
Redis
Message Handler


Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
Message Service
Group Service
WebSocket Connection Manager
kafka
cassandra
Redis
Message Handler


Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
Message Service
Group Service
WebSocket Connection Manager
kafka
cassandra
CACHE
Redis
Message Handler

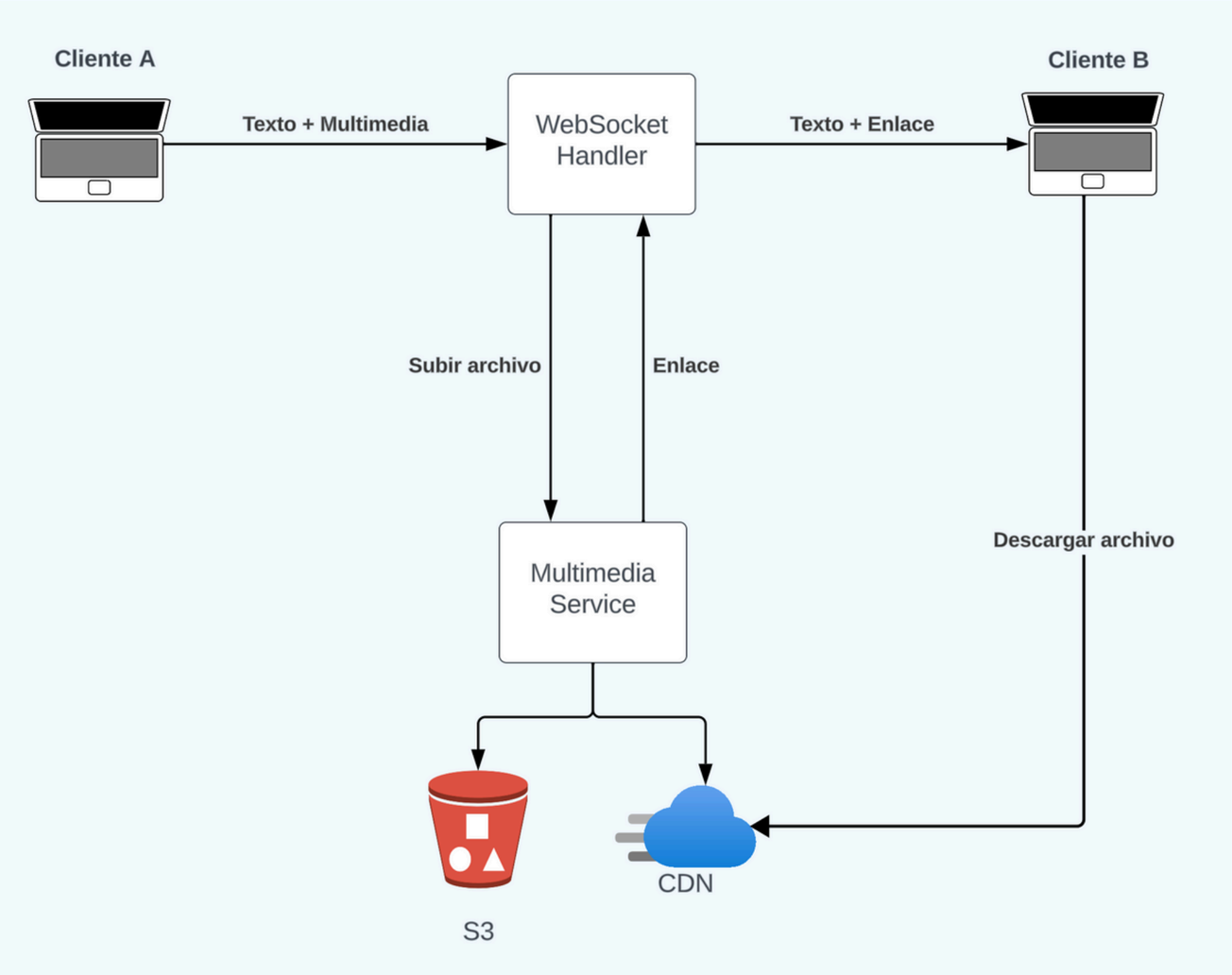

Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
Message Service
Group Service
WebSocket Connection Manager
Multimedia Service
kafka
cassandra
CACHE
Redis
S3
CDN
Message Handler

ARCHIVOS MULTIMEDIA

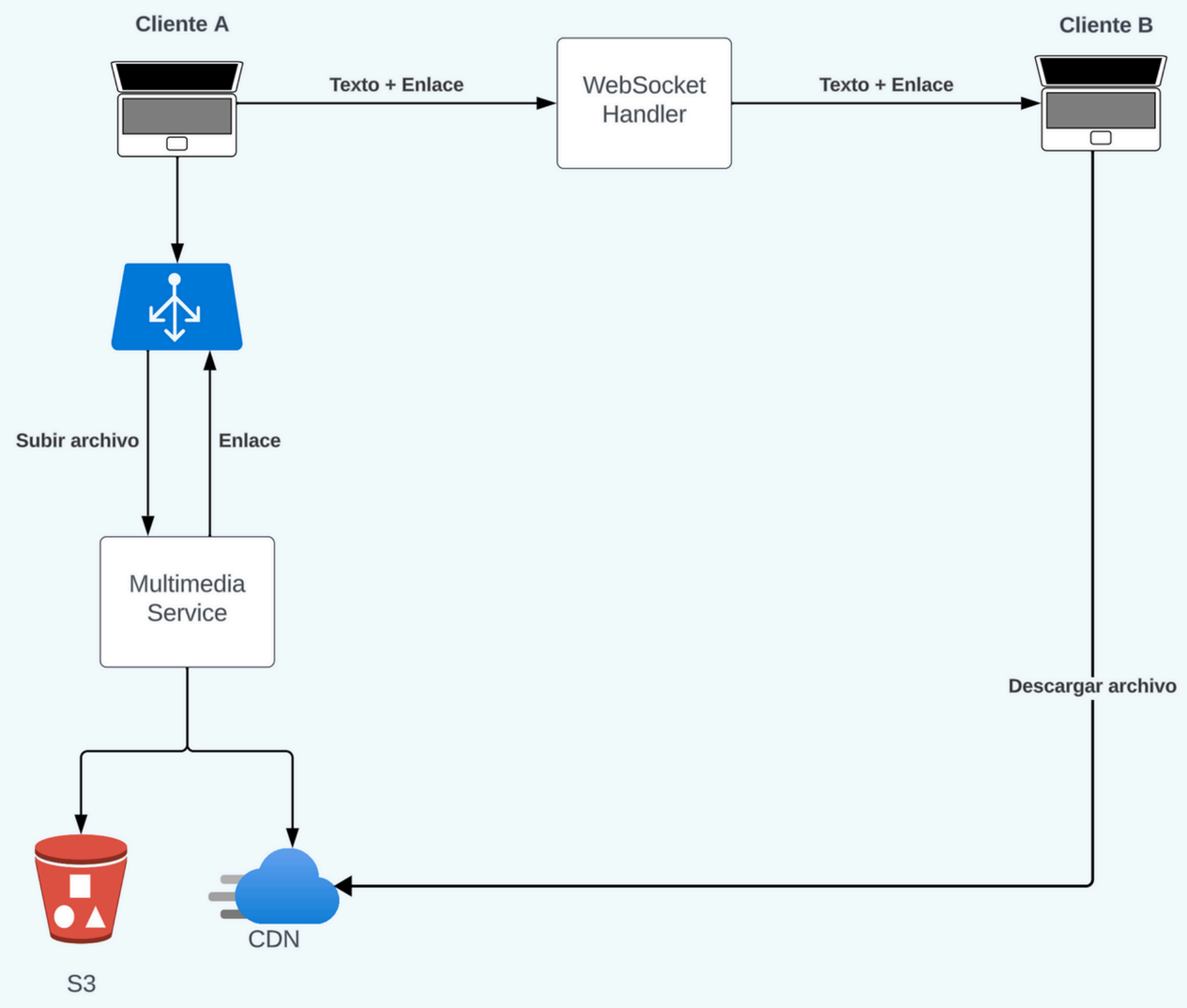
- **Compresión de archivos** en el lado del cliente. Límite en el tamaño.

- No nos podemos fiar de los clientes.
  - Implementar **límite en el lado del servidor.**
  - Comprimir archivos si no lo ha hecho el cliente.

ARCHIVOS MULTIMEDIA



ARCHIVOS MULTIMEDIA

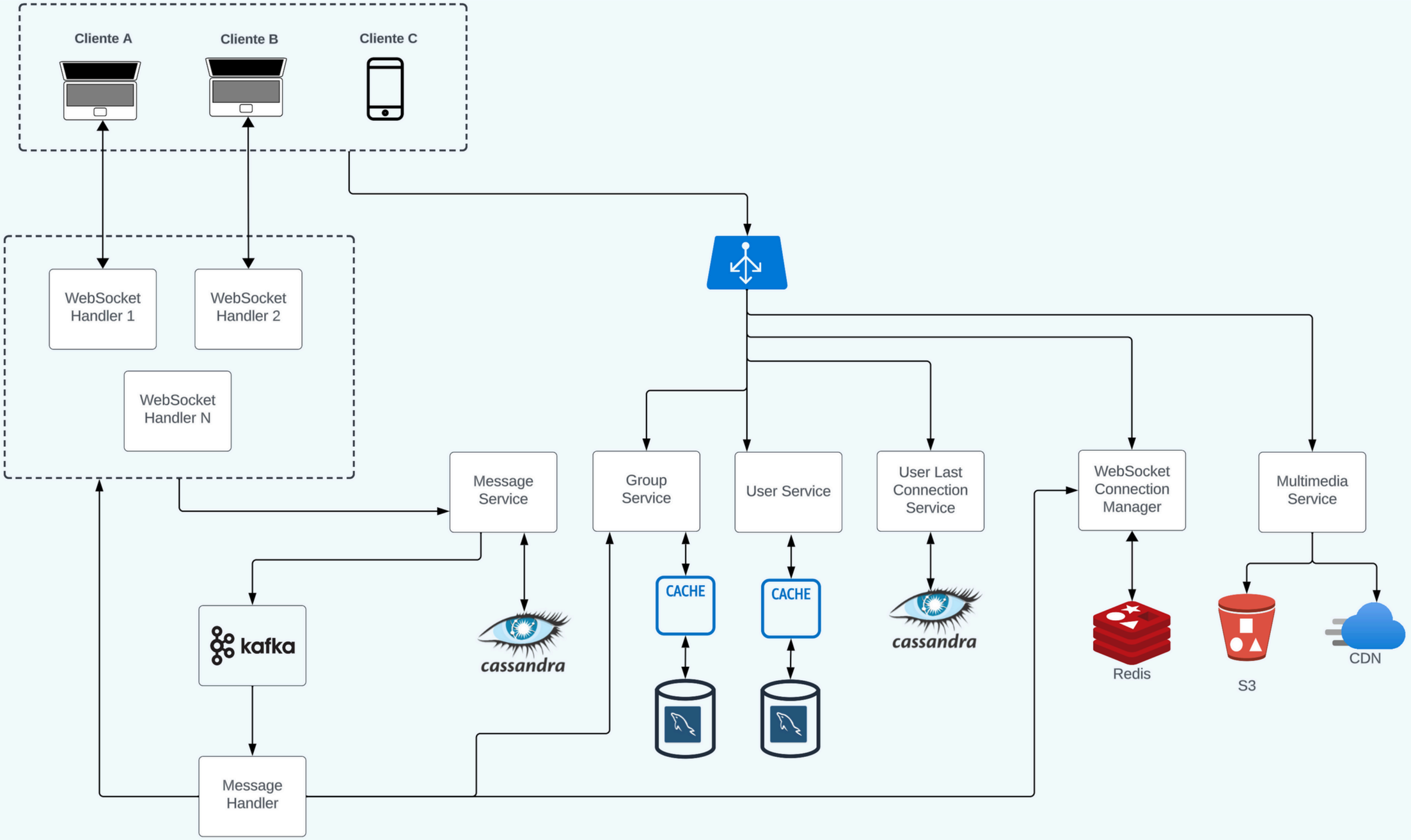


ARCHIVOS MULTIMEDIA

- **Compresión de archivos** en el lado del cliente. Límite en el tamaño.

- No nos podemos fiar de los clientes.
  - Implementar **límite en el lado del servidor.**
  - Comprimir archivos si no lo ha hecho el cliente.

- **Detección de ficheros duplicados** en el multimedia service en el cliente y en el servidor.
  - Los clientes preguntarán al multimedia service si ya tiene ese fichero antes de subirlo.
  - El multimedia service realizará también dicha comprobación al recibir el fichero.
  - Se aplican **múltiples algoritmos hash** para evitar problemas de colisiones.


Cliente A
Cliente B
Cliente C
WebSocket Handler 1
WebSocket Handler 2
WebSocket Handler N
Message Service
Group Service
User Service
User Last Connection Service
WebSocket Connection Manager
Multimedia Service
kafka
cassandra
CACHE
CACHE
cassandra
Redis
S3
CDN
Message Handler

MODELO DE DATOS

direct_message
message_id {PK}
from_user_id
to_user_id
content
created_at

group_message
message_id {PK}
group_id {PK}
from_user_id
content
created_at

MODELO DE DATOS

direct_message
message_id {PK}
from_user_id
to_user_id
content
created_at

group_message
message_id {PK}
group_id {PK}
from_user_id
content
created_at

- Los message_id deben ser **únicos y ordenables por tiempo.**
  - created_at no puede ser id. Dos mensajes se pueden enviar al mismo tiempo.
  - Generador global de identificadores únicos.
  - El id puede ser único de forma local, para cada chat o grupo.
    - Podría ser de la forma **${USER_ID}${DESTINATION_ID}${LOCAL_SEQUENCE}**.
      - Podríamos usar created_at para ordenar los mensajes.

OPTIMIZACIONES EN LOS CLIENTES

- Búsqueda de mensajes.
  - Solicitar de nuevo los mensajes al servidor malgastaría recursos.

- Debemos guardar los mensajes de forma local en los clientes.
  - Para Android / iOS podemos utilizar una BBDD como SQLite.
  - Para una web podemos utilizar WebStorage.
    - Límite de 5MB. Es probable que tengamos que hacer llamadas al servidor.
    - Existen otras alternativas como almacenar ficheros en el sistema de ficheros del usuario, pero no son estándar.

- Envío de mensajes sin conexión.
  - Se almacenan con estado no enviado hasta recuperar la conexión a internet.