

# Poc Desarrollo

## Participantes:

- Lautaro Lier
- Constantino Finelli
- Luciano Acosta

---

## Protocolo

## Protocolos previos a Websockets

### short polling

El **short polling** es una técnica utilizada en la comunicación cliente-servidor donde un cliente envía repetidamente solicitudes a un servidor en intervalos cortos y regulares para verificar si hay actualizaciones o datos nuevos.

Se llama *short polling* porque el intervalo entre cada solicitud es relativamente breve.

---

### Caso de uso

Se utiliza comúnmente en escenarios donde las actualizaciones en tiempo real **no son críticas**.

---

### ¿Cómo funciona el short polling?

1. **Inicio de la solicitud:** el cliente envía una petición al servidor.
2. **Respuesta inmediata:** el servidor procesa la petición y responde enseguida, aunque no haya datos nuevos.
3. **Repetición de solicitudes:** el cliente espera un intervalo de tiempo predeterminado y vuelve a enviar otra petición.

---

### Desventajas

- **Ineficiente para necesidades en tiempo real:** la alta frecuencia de solicitudes puede generar tráfico de red innecesario y sobrecargar al servidor si las actualizaciones son poco frecuentes, ya que el cliente consulta todo el tiempo aunque no haya novedades.
- **Latencia:** el cliente no recibe los datos de inmediato, depende del intervalo configurado.
- **Problemas de escalabilidad:** a medida que crece el número de usuarios, también aumenta la cantidad de solicitudes de red, lo que agrava el problema anterior.

### Long Polling

El **long polling** es una técnica de comunicación tipo *server push* utilizada en el backend para emular interacción en tiempo real sobre HTTP, normalmente en aplicaciones web que no tienen soporte para WebSockets o HTTP/2.

El cliente envía una solicitud al servidor, que mantiene la conexión abierta hasta que haya datos nuevos disponibles o se cumpla un tiempo de espera (*timeout*). Una vez que el servidor responde, el cliente inmediatamente vuelve a iniciar otra solicitud, manteniendo así un flujo de datos casi en tiempo real.

Al mantener la conexión abierta entre cliente y servidor, el long polling reduce la frecuencia percibida de peticiones, lo que lo convierte en una buena opción para **apps de chat, juegos multijugador y sistemas en vivo**.

Sin embargo, a diferencia de WebSockets, Server-Sent Events (SSE) o *streams* de gRPC, sigue teniendo el problema del **overhead HTTP repetido**, lo que puede volverse costoso a gran escala si no se configura correctamente.

## Long Polling vs Short Polling

- **Short Polling (pull-based):** el cliente envía peticiones periódicas y el servidor responde enseguida, incluso si no hay datos. Es ineficiente en tiempo real porque genera mucho tráfico de red innecesario y mayor latencia.
- **Long Polling (push-based):** el servidor mantiene abierta la petición hasta que tenga novedades, lo que permite entregar los datos en cuanto aparecen y reducir peticiones vacías.

## Server-Sent Events (SSE)

Los **Server-Sent Events (SSE)** son una tecnología de red que permite a los servidores **enviar actualizaciones en tiempo real a los clientes** a través de una única conexión HTTP.

Al igual que otros enfoques de *data streaming* en la web, los SSE establecen una **conexión HTTP de larga duración** entre el servidor y el cliente.

- Una vez establecida, el servidor puede **enviar datos al cliente en cualquier momento**, sin que el cliente tenga que hacer solicitudes adicionales.
- Esto contrasta con las conexiones HTTP tradicionales, donde el cliente debe consultar constantemente al servidor (*polling*).

---

## ¿Cómo funciona una aplicación con SSE?

### Flujo de conexión:

1. El cliente hace una **solicitud HTTP GET** al endpoint SSE.
2. El servidor responde con **200 OK** y el header `Content-Type: text/event-stream`.
3. A partir de ahí, el servidor puede enviar eventos continuamente.

### Formato de un evento SSE:

```
event: nombreEvento  
data: datosDelEvento
```

- `event:` → el tipo de evento.
- `data:` → la información enviada (ej. JSON, texto, etc.).

El cliente escucha estos eventos con JavaScript (por ejemplo, usando `EventSource`) y los procesa en cuanto llegan.

La conexión SSE **permanece abierta** hasta que el cliente o servidor la cierren.

- Si el cliente cierra, ya no recibe datos.
- Si el servidor cierra, el cliente intentará **reconectarse automáticamente**.

## Ventajas de usar SSE

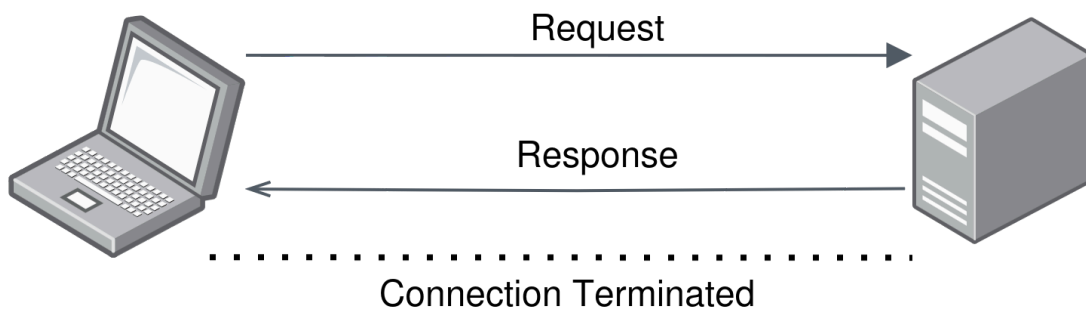
1. **Unidireccional simple:** ideal para *streaming* de datos **del servidor al cliente** (ej: cotizaciones bursátiles, precios de Bitcoin, notificaciones en vivo).
2. **Simplicidad y facilidad de uso:** el cliente se suscribe a un endpoint SSE, y el servidor empuja los datos sin necesidad de solicitudes repetitivas.
3. **Menor overhead de red:** en comparación con *polling*, el servidor solo envía datos cuando hay algo nuevo.
4. **Protocolo estandarizado:** funciona sobre HTTP normal, fácil de desplegar en la infraestructura web existente.
5. **Reconexión automática:** si se corta la conexión, el cliente intenta reconectarse solo.
6. **Soporte de cross-domain:** puede funcionar entre distintos dominios, lo que da flexibilidad en el diseño de aplicaciones.
7. **Compatibilidad y accesibilidad:** basado en HTML5, ampliamente soportado por navegadores modernos, sin necesidad de plugins o software extra.

## ¿Que es/son los websockets y como funciona?

WebSockets, o WS para mayor facilidad. Es un protocolo de comunicación **bidireccional**, **persistente** y **a tiempo real**, por medio de una **única conexión TCP\***.

\*TCP, o **Protocolo de Control de Transmisión**, es un protocolo fundamental en las redes de computadoras que garantiza la entrega confiable de datos entre dispositivos.

Normalmente la comunicación entre cliente y servidor se basa en una serie de peticiones y respuestas, ya sea para enviar o solicitar información al servidor y que el mismo realiza una respuesta



Este esquema para algunos tipos de aplicaciones, como mensajería u otras aplicaciones las cuales dependían de una actualización de datos constantes tiene los siguientes problemas:

- El servidor era forzado a establecer muchas conexiones TCP para cada cliente. En el caso de una aplicación de mensajera, el servidor tiene que manejar las conexiones hechas para sincronizar los mensajes de cada usuario. Es decir recibir el mensaje enviado por uno y enviarlo al otro.  
Con el esquema de peticiones y respuestas, para subir y recibir cada mensaje, era necesario hacer una conexión TCP.
- A su vez, cada conexión, a nivel de red, se encontraba sobrecargada ya que cada mensaje del cliente al servidor incluye un encabezado HTTP.
- El cliente se ve obligado a mantener una correspondencia entre las conexiones salientes y las conexiones entrantes para poder rastrear las respuestas.

Gracias a los WebSockets estos problemas se resuelven, ya que solo se establece una conexión entre el servidor y el cliente por donde se hará la comunicación.

Primeramente el cliente realiza una petición al servidor, solicitando el establecimiento de la conexión. El servidor responde, y si es afirmativa la conexión se establece el puente para la comunicación y transferencia de datos de forma bidireccional.

Ambos, el cliente y el servidor, pueden enviar y recibir mensajes de manera independiente. Significa que ninguno está activamente esperando a que llegue un mensaje para poder realizar una acción.



El handshake es el proceso inicial de establecer una conexión de WebSocket entre un cliente y un servidor.

---

## Flujo de un WebSocket

1. El cliente envía una petición HTTP Upgrade (**es un mecanismo en HTTP/1.1 que permite cambiar a un protocolo diferente (o una versión más nueva) durante una conexión existente, funciona como un GET especial**)
  2. El servidor acepta y actualiza a protocolo WebSocket
  3. Se establece una conexión con persistencia
  4. Los mensajes ahora pueden enviarse en ambas direcciones con una sobrecarga mínima
- 

## En donde se puede aplicar ?

El protocolo WS es altamente usado hoy en día. Principalmente en juegos y aplicaciones de mensajería, como lo pueden ser WhatsApp y Telegram.

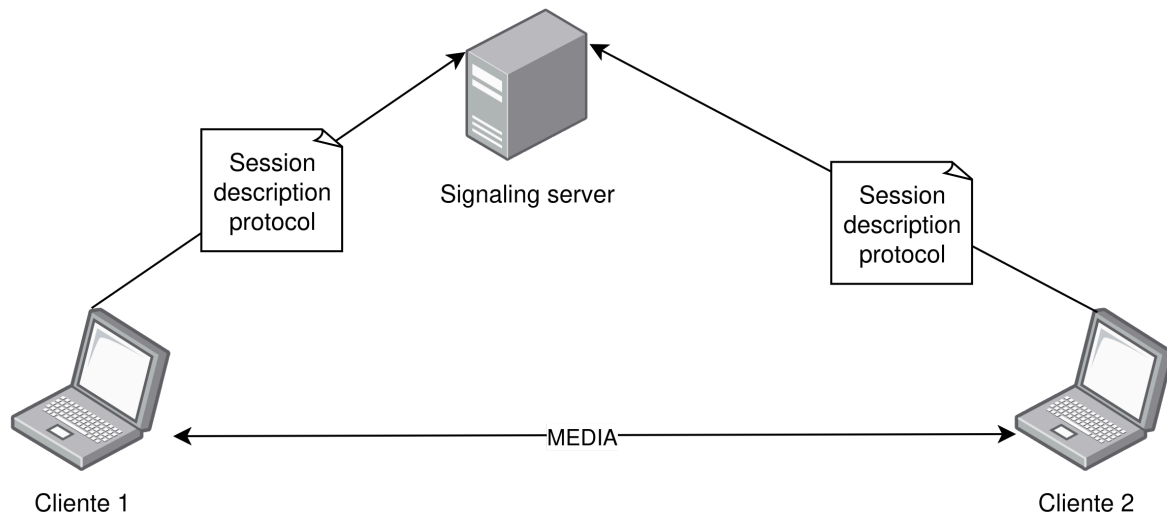
Algunos servicios utilizan este método para poder brindar datos de forma actualizada. Como lo pueden ser aplicaciones/servicios de finanzas o criptomonedas.

Otros ejemplos de uso de esta tecnología pueden ser la edición colaborativa de documentos (ej. Google Docs).

---

## ¿Qué es WebRTC y cómo funciona?

WebRTC (Web Real-Time Communication) es una tecnología y conjunto de protocolos que permite la comunicación **en tiempo real** directamente entre navegadores o aplicaciones, sin necesidad de un servidor intermedio para la transmisión de datos (aunque sí requiere uno para la *señalización inicial*).



La mayoría de los dispositivos (celulares, computadoras ,etc) se encuentran detrás de un firewall o sus IPs cambian contentamente. Es por esto que se utiliza un estándar conocido como: Interactive Connectivity Establishment (ICE).

Este estándar facilita la coordinación entre clientes para poder establecer conexiones entre ellos. Cada cliente genera un “reporte” o ICE candidate, en donde se indica la IP y puerto que se puede usar para establecer una conexión. Estos luego son enviados a uno o mas STUN (Session Transversal Utilities for Nat) servers.

Los STUN servers pueden ser consultados para poder establecer las IPs y puertos para poder generar los Session Description Protocols que se envían al Signaling Server para poder establecer la conexión.

Está optimizado para **audio, video y datos** con **baja latencia**, y utiliza principalmente protocolos basados en UDP para mejorar la velocidad y reducir retrasos.

Para funcionar, WebRTC emplea:

- **STUN**: Determina la IP pública y el puerto accesible del cliente.
- **TURN**: Reenvía datos cuando no es posible la conexión directa por restricciones de red (NAT estrictos).
- **SRTP/DTLS**: Proporcionan encriptación y seguridad en la transmisión.

## Comparación WebSockets vs WebRTC

Característica	WebSockets	WebRTC
<b>Tipo de conexión</b>	Cliente-Servidor	Peer-to-Peer (con señalización)
<b>Protocolo base</b>	TCP (ws/wss)	UDP (SRTP, DTLS)
<b>Optimizado para</b>	Datos en tiempo real (texto, binario)	Audio, video y datos

Característica	WebSockets	WebRTC
<b>Latencia</b>	Baja (50-200 MS)	Muy baja (<50 MS)
<b>Servidor necesario</b>	Siempre	Solo para señalización / relay
<b>Complejidad de implementación</b>	Baja	Alta
<b>Casos de uso típicos</b>	Chats, notificaciones, juegos online simples	Videollamadas, streaming, compartición de pantalla


## Conclusión

- **WebSockets:** ideales para datos en tiempo real de cualquier tipo, especialmente cuando hay un servidor central que gestiona la comunicación.
- **WebRTC:** la opción preferida para audio/video en vivo y datos con latencia mínima, aprovechando conexiones P2P para optimizar el rendimiento.

## Referencias

### RFC 6455: The WebSocket Protocol


The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is

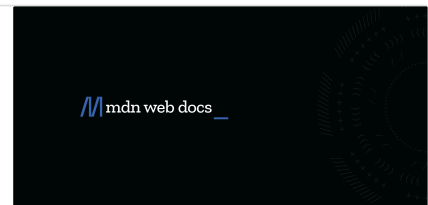
 <https://datatracker.ietf.org/doc/html/rfc6455>



### The WebSocket API (WebSockets) - Web APIs | MDN

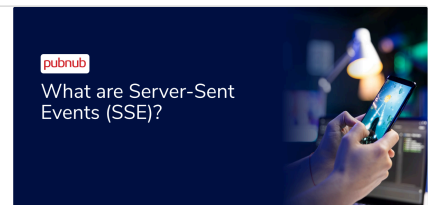
The WebSocket API makes it possible to open a two-way interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive responses without having to poll

 [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)



### What are Server-Sent Events (SSE)?

 <https://www.pubnub.com/guides/server-sent-events/>




### What is Long Polling and How Does it Work?

 <https://www.pubnub.com/guides/long-polling/>



### Understanding Short Polling: A Simple Approach to Client-Server Communication

What is a short polling? Short polling is a technique used in client-server communication where a...

 <https://dev.to/simranjit884/understanding-short-polling-a-simple-approach-to-client-server-communication-3ldm>



### API de WebRTC - API web | MDN


WebRTC (Web real-time communications o comunicaciones web en tiempo real) es una tecnología que permite a aplicaciones y sitios web capturar y opcionalmente retransmitir audio/video, así como intercambiar datos arbitrarios

 [https://developer.mozilla.org/es/docs/Web/API/WebRTC\\_API](https://developer.mozilla.org/es/docs/Web/API/WebRTC_API)



### Comienza a usar WebRTC

Crear una aplicación nueva basada en las tecnologías WebRTC puede ser abrumador si no estás familiarizado con las API. En esta sección, te mostraremos cómo comenzar con las diversas API del estándar WebRTC, lo que explicará una serie de casos de uso comunes y fragmentos de código para resolver esos problemas.

 <https://webrtc.org/getting-started/overview?hl=es-419>