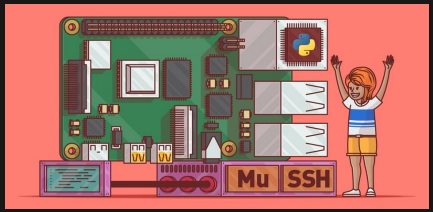
Protocolos de comunicación entre dispositivos: WebSockets

Introducción al Internet de las Cosas



Marco Teran
Universidad Sergio Arboleda

2023

Contenido

Comunicación cliente-servidor

Comunicación cliente-servidor

Comunicación cliente-servidor

La comunicación cliente-servidor es una estructura de aplicación distribuida que divide tareas o cargas de trabajo entre proveedores de recursos o servicios, llamados **servidores**, y solicitantes de servicio, llamados **clientes**.

Características principales

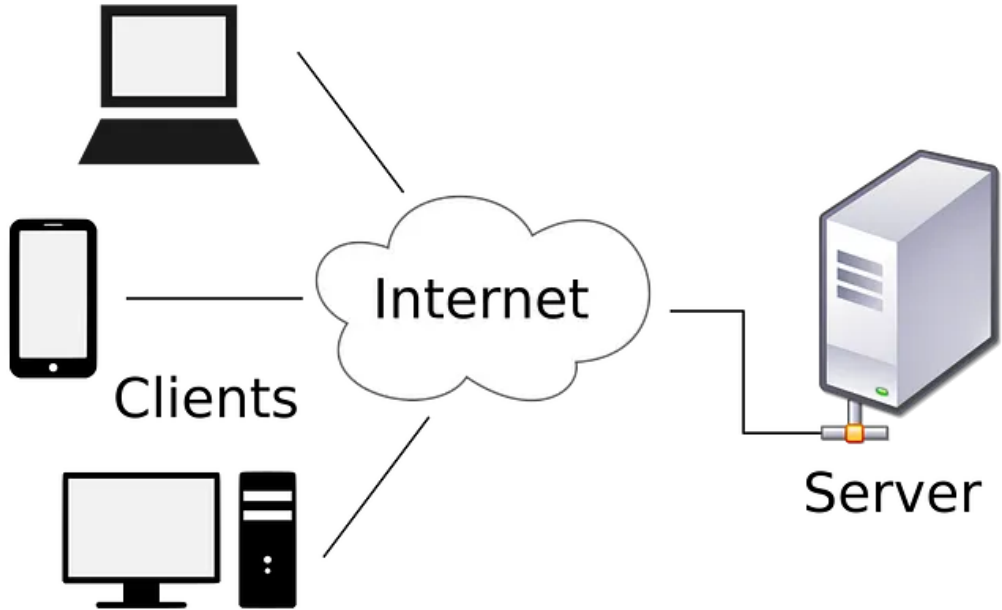
- La característica **cliente-servidor** describe la relación de programas cooperantes en una aplicación
- La comunicación es a través de una red.
- La comunicación entre el cliente y el servidor sigue un patrón de intercambio de mensajes de solicitud-respuesta.
- Los protocolos de comunicación definen las reglas y el lenguaje de la comunicación.

Características principales

- Los servidores comparten sus recursos con los clientes.
- El servidor proporciona servicios a uno o muchos clientes.
- Un recurso compartido puede ser cualquier software o componente electrónico del servidor.
- Los servidores se clasifican por los servicios que proporcionan, como servidores web o de archivos.
- Los servidores esperan solicitudes entrantes.

Características principales

- El componente servidor proporciona una función o servicio a uno o muchos clientes, que inician solicitudes de dichos servicios.
- Los clientes no comparten sus recursos.
- Los clientes inician sesiones de comunicación con los servidores.


Internet
Clients
Server

Client-host y server-host

La host

Terminales que están conectados a una red. A diferencia de los términos cliente y servidor, que pueden referirse a una computadora o un programa de computadora, client-host y server-host siempre se refieren a una terminal en específico. La host es una terminal multifuncional, mientras que los clientes y servidores son programas que se ejecutan en una host.

- Host se refiere a cualquier terminal conectado a una red.
- Clientes y servidores son programas que se ejecutan en una host.

Arquitecturas de la comunicación cliente-servidor

Computación centralizada

Centralized computing

Modelo que asigna una gran cantidad de recursos a un número limitado de terminales. Los cálculos se realizan principalmente en el terminal central, mientras que los terminales clientes tienen una menor carga computacional.

Características principales:

- Cálculos se realizan en la terminal central.
- Los terminales clientes tienen una menor carga computacional.

Arquitectura de pares (peer-to-peer)

Peer-to-peer architecture

Modelo en el que dos o más terminales comparten recursos y se comunican en un sistema descentralizado. Los nodos pares son coiguales y no hay una jerarquía de red.

- Dos o más terminales (pares) comparten recursos y se comunican en un sistema descentralizado.
- Los pares son nodos co-iguales en una red no jerárquica.

Características de la red de pares (peer-to-peer)

- A diferencia de las redes cliente-servidor o cliente-cola-cliente, los pares se comunican directamente entre sí.
- Un algoritmo en el protocolo de comunicaciones de pares equilibra la carga, y los pares modestos pueden ayudar a compartir la carga.
- Si un nodo se vuelve no disponible, sus recursos compartidos siguen disponibles siempre que otros pares lo ofrezcan.
- Idealmente, un par no necesita lograr alta disponibilidad porque otros pares redundantes compensan cualquier tiempo de inactividad de recursos.
- Tanto cliente-servidor como maestro-esclavo son subcategorías de sistemas de pares distribuidos.
- El protocolo reenruta solicitudes a medida que la disponibilidad y capacidad de carga de los pares cambia.

Tipos de clientes

Tipos de clientes

Tipos de clientes

Un cliente es un dispositivo o software de servidor que solicita recursos y servicios disponibles a través de un servidor. El cómputo del cliente se clasifica como Grueso, Delgado o Híbrido.

- Cliente Grueso (Thick Client): proporciona una funcionalidad rica, realiza la mayor parte del procesamiento de datos por sí mismo y depende muy poco del servidor.
- Cliente Delgado (Thin Client): un servidor cliente ligero es una computadora liviana que depende en gran medida de los recursos de la computadora anfitriona. El servidor de aplicaciones realiza la mayor parte del procesamiento de datos requerido.
- Cliente Híbrido (Hybrid Client): posee una combinación de características del cliente delgado y del cliente grueso, depende del servidor para almacenar datos persistentes pero es capaz de procesamiento local.

Tipos de Servidores

Tipos de Servidores

Tipos de Servidores

Un servidor es un dispositivo o programa informático que proporciona funcionalidad para otros dispositivos o programas. Cualquier proceso informático que pueda ser utilizado o solicitado por un cliente para compartir recursos y distribuir trabajo es un servidor.

Tipos de Servidores

Tipos de Servidores

Algunos ejemplos comunes de servidores incluyen:

- Servidores de Aplicaciones (Application Server): alojan aplicaciones web para su uso en la red sin necesidad de copias locales
- Servidores de Cómputo (Computing Server): comparten recursos informáticos con otras computadoras en red que necesitan mayor potencia de CPU y RAM que la disponible en una computadora personal.
- Servidores de Base de Datos (Database Server): mantienen y comparten bases de datos para programas informáticos que procesan datos bien organizados, como software de contabilidad y hojas de cálculo.
- Servidores Web (Web Server): alojan páginas web y hacen posible la existencia de la World Wide Web.

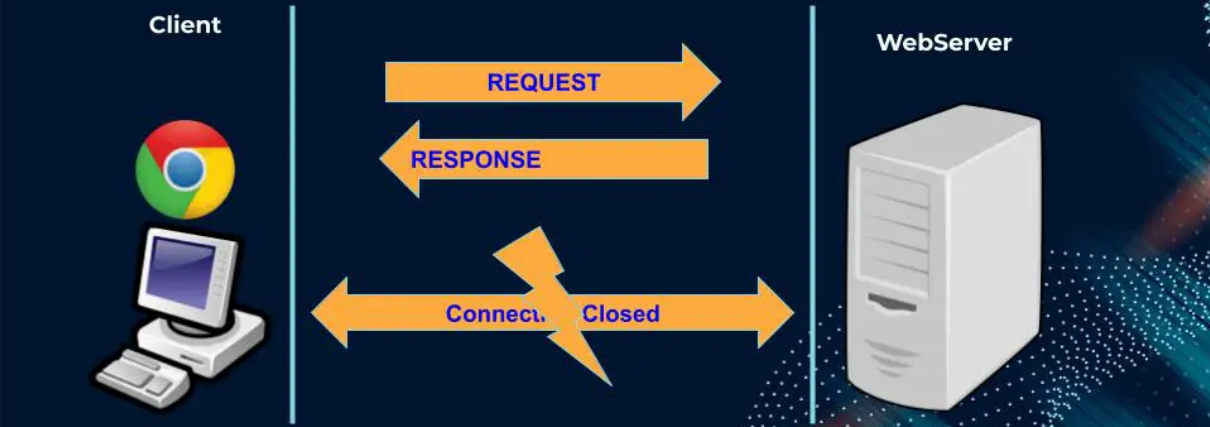
Protocolo HTTP

Introducción al Protocolo HTTP

El Protocolo de Transferencia de Hipertexto (HTTP) es el protocolo más utilizado para la comunicación en la web y en el Internet de las Cosas (IoT).

- La mayoría de la comunicación en la web se realiza a través de HTTP.
- Hay dos modos de transferencia de datos entre el cliente y el servidor: **HTTP PUSH** y HTTP PULL.
- En **HTTP PULL**, el cliente envía una solicitud y el servidor responde con los datos.
- En **HTTP PUSH**, el servidor envía actualizaciones al cliente cuando están disponibles.
- Existen múltiples tecnologías involucradas en el mecanismo basado en **HTTP PUSH**.

HTTP



Tecnologías de HTTP PUSH

El mecanismo de HTTP PUSH involucra múltiples tecnologías como:

- Ajax Long polling
- Web Sockets
- HTML5 Event Source (Server-sent event)
- Message Queues
- Streaming over HTTP

Long-Polling, WebSockets y Server-Sent Events son los protocolos de comunicación más populares entre el cliente y el servidor.

Características del Protocolo HTTP

Características del Protocolo HTTP

El Protocolo HTTP es síncrono en el sentido de que cada solicitud recibe una respuesta, pero asincrónico en el sentido de que las solicitudes pueden tardar mucho tiempo y múltiples solicitudes pueden ser procesadas en paralelo.

- Por lo tanto, muchos clientes y servidores HTTP se implementan de manera asincrónica, pero sin ofrecer una API asincrónica.
- La implementación asincrónica puede realizarse mediante múltiples hilos, callbacks o bucles de eventos.
- La versión más utilizada actualmente es HTTP/1.1.
- El protocolo es ampliamente utilizado para dispositivos IoT, aunque no es el más adecuado debido a sus limitaciones.

Limitaciones del protocolo HTTP en IoT

- **Diseñado para comunicación de dos sistemas:** HTTP fue diseñado originalmente para la comunicación de dos sistemas a la vez, lo que lo hace ineficiente y consume mucho tiempo y energía para conectar varios sensores para obtener información.
- **Unidireccional:** El protocolo HTTP es unidireccional, lo que significa que está diseñado para que un sistema (cliente) envíe un mensaje a otro (servidor). Esto hace que sea difícil escalar soluciones de IoT.
- **Consumo de energía:** HTTP utiliza el Protocolo de Control de Transmisión (TCP), que requiere una gran cantidad de recursos informáticos y no es adecuado para aplicaciones alimentadas por batería.

Limitaciones del protocolo HTTP en IoT

- Es importante considerar las limitaciones de los protocolos de comunicación para soluciones de IoT.
- Existen alternativas específicas para IoT, como WebSockets, CoAP, MQTT y AMQP.
- En la comunicación web, existen dos protocolos importantes: HTTP y Websocket.
- La elección del protocolo adecuado es fundamental para garantizar una comunicación eficiente, segura y confiable entre los dispositivos IoT y el servidor.

WebSockets

WebSockets

WebSockets

WebSockets es una tecnología que permite una comunicación bidireccional en tiempo real entre un cliente y un servidor a través de una única conexión TCP (Transfer Control Protocol, *ing.* Protocolo de Control de Transmisión). Es una tecnología de comunicación en tiempo real entre el cliente y el servidor a través de una conexión persistente y bidireccional.

WebSockets fueron inventados por el Consorcio World Wide Web (W3C) en 2008 para aprovechar las ventajas de HTTP en la capa de aplicación y ofrecer la riqueza del TCP en la capa de transporte (Modelo OSI).

Características del WebSocket

Características de los WebSockets

- TCP establece y mantiene conexiones entre hosts hasta que las aplicaciones en cada extremo han terminado de intercambiar mensajes, facilitando el transporte de solicitudes y respuestas HTTP.
- Las aplicaciones web pueden iniciar una conexión similar a TCP a través de una solicitud HTTP inicial y luego .ªctualizar.ᵉˢta conexión a un WebSocket.
- La conexión TCP / IP existente ahora se puede usar como una conexión WebSocket, a través de la cual los datos pueden fluir en ambas direcciones, fuera del mecanismo de solicitud / respuesta HTTP.
- Se utiliza una única conexión TCP para la comunicación.
- El uso de WebSockets permite una transferencia de datos en tiempo real y una baja sobrecarga de comunicación.

Características de los WebSockets

- Comunicación bidireccional: WebSockets son canales de comunicación persistentes full-duplex sobre una única conexión TCP.
- Tanto el servidor como el cliente pueden enviar datos en cualquier momento, gracias al proceso conocido como "WebSocket handshake".
- Con una sola conexión TCP se pueden enviar múltiples mensajes entre un cliente y un servidor mediante un único handshake HTTP.
- Conexión persistente: El servidor puede enviar contenido al navegador sin ser solicitado por el cliente, lo que permite pasar mensajes en ambas direcciones mientras se mantiene abierta la conexión. La conexión se mantiene abierta mientras sea necesario. Lo que significa que no se crea ni destruye en cada ciclo de solicitud y respuesta, lo que reduce la sobrecarga en la red.

Características de los WebSockets

- WebSockets son una buena opción para aplicaciones que involucren múltiples usuarios comunicándose entre sí, o que den acceso a datos del servidor que cambian constantemente.
- Algunos casos de uso incluyen redes sociales, edición/codificación colaborativa, aplicaciones de chat, seguimiento de usuarios, interacción en vivo con la audiencia y actualizaciones de dispositivos IoT.

Características de la conexión WebSocket

- Transferencia de datos en formato binario o texto.
- Eficiencia: la conexión WebSocket tiene un menor uso de recursos en comparación con otras tecnologías de comunicación en tiempo real.
- Seguridad: la conexión WebSocket utiliza SSL/TLS para garantizar la seguridad de la comunicación.
- Es más eficiente que otras alternativas como el polling.
- Estas capacidades habilitan aplicaciones en juegos, redes sociales, logística, finanzas, hogar, vehículos y automatización industrial, entre otros.

Caracteristicas del WebSocket

- WebSocket es compatible con varios lenguajes de programación, como JavaScript, Python, Java, C ++ y muchos más.
- WebSocket es capaz de manejar grandes volúmenes de datos en tiempo real, lo que lo hace ideal para aplicaciones IoT que requieren la transmisión de datos a alta velocidad.
- WebSocket utiliza un protocolo de enlace de extremo a extremo, lo que significa que los datos se pueden enviar directamente entre los clientes y el servidor sin intermediarios, lo que reduce la latencia y mejora la eficiencia de la transmisión de datos.

Conexión WebSocket

Conexión WebSocket

Proceso de conexión:
El proceso de conexión WebSocket consta de los siguientes pasos:

1. El cliente envía una solicitud con la cabecera Upgrade para cambiar al protocolo WebSocket.
2. El servidor recibe la solicitud, la procesa y decide si puede establecer una conexión WebSocket.
3. Si el servidor puede establecer la conexión, envía una respuesta al cliente para confirmar la conexión WebSocket.
4. Si el servidor no puede establecer la conexión, envía una respuesta para informar al cliente.
5. Una vez que se establece la conexión, el cliente y el servidor pueden enviar datos en ambas direcciones.
6. La conexión se cierra cuando el servidor o el cliente decide cerrarla. Si uno de los participantes se desconecta, se cierra la conexión sin posibilidad de reconexión.

Tipos de conexión WebSocket

De acuerdo a la dirección de la solicitud:

1. Cliente a servidor: el cliente inicia la conexión y envía solicitudes al servidor.
2. Servidor a servidor: el servidor inicia la conexión con otro servidor para intercambiar datos.

De acuerdo al número de canales utilizados:

- Full-duplex: Permite la comunicación bidireccional simultánea de datos entre un cliente y un servidor.
- Half-duplex: Permite la comunicación en ambas direcciones, pero solo en un momento dado.

El protocolo WebSocket

El protocolo WebSocket

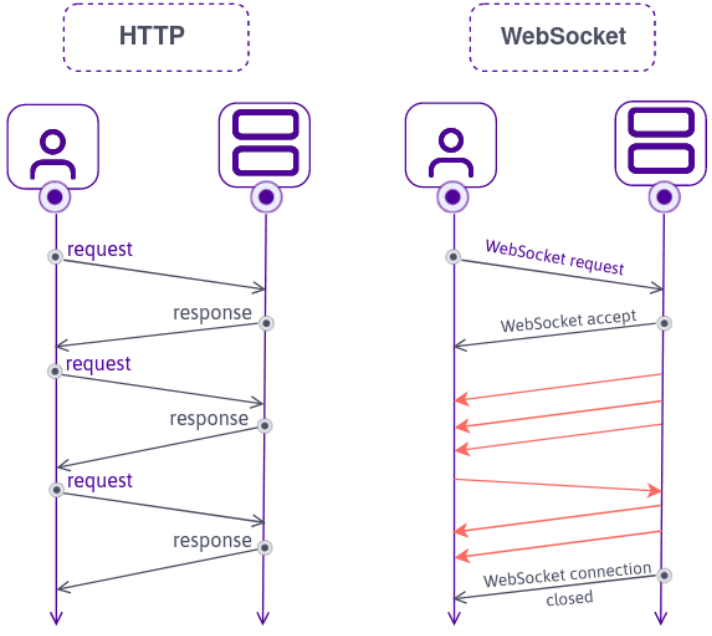
El protocolo WebSocket

El protocolo WebSocket se creó para permitir que las aplicaciones web realicen comunicaciones bidireccionales con los servidores web, ya que HTTP no puede hacerlo sin soluciones incómodas. Está definido en el RFC 6455.

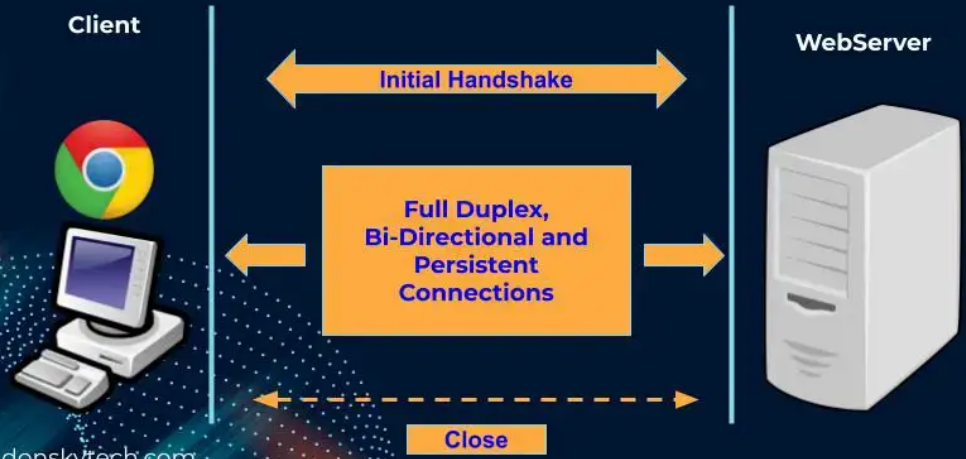
- Un WebSocket es una conexión única de larga duración (persistente), de baja latencia (en tiempo real) y dúplex completo (bidireccional) entre un cliente y un servidor.

WebSocket Protocolo de comunicación

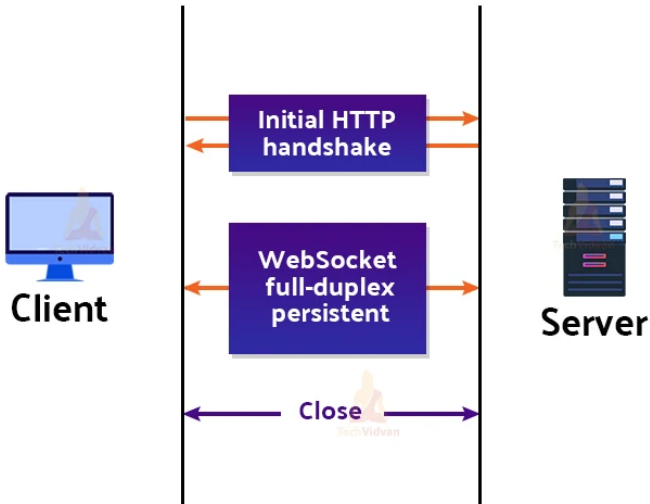
- Diferente de HTTP pero compatible con HTTP.
- Localizado en la capa 7 del modelo OSI y depende de TCP en la capa 4.
- Funciona a través de los puertos 80 y 443 (en caso de encriptación TLS) y admite intermediarios HTTP.

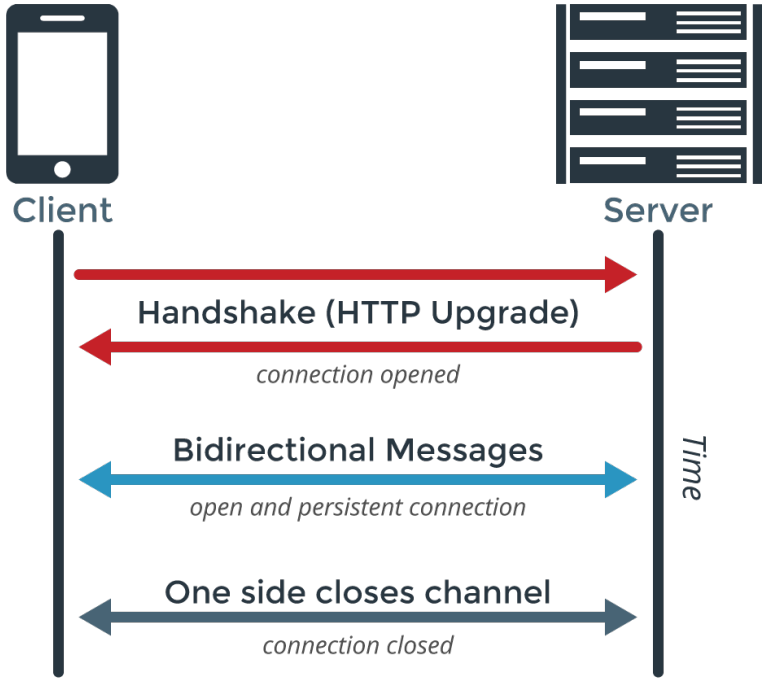

HTTP
request
response
request
response
request
response
WebSocket
WebSocket request
WebSocket accept
WebSocket connection
closed

WebSocket


Client
WebServer
Initial Handshake
Full Duplex,
Bi-Directional and
Persistent
Connections
Close

WebSocket


Initial HTTP handshake
WebSocket full-duplex persistent
Close
Client
Server


Client
Server
Handshake (HTTP Upgrade)
connection opened
Bidirectional Messages
open and persistent connection
One side closes channel
connection closed
Time

Aplicaciones del WebSocket

WebSocket en proyectos de IoT

- WebSocket es una excelente forma de comunicación para mostrar datos en tiempo real que se envían desde el servidor a los clientes.
- Es ideal para aplicaciones en tiempo real como aplicaciones de chat, precios de acciones y programas de juegos que requieren actualizaciones asincrónicas en tiempo real.
- WebSocket es muy aplicable a proyectos basados en IoT, ya que se utilizan sensores o actuadores que necesitan una respuesta inmediata.
- La mayoría de los proyectos IoT requieren control desde navegadores web, y la mayoría de los navegadores admiten el protocolo WebSocket.

WebSocket en proyectos de IoT

- Un ejemplo típico es cuando se controla un robot de coche WIFI usando un teléfono móvil. La aplicación web que se utiliza para controlar el robot debe responder inmediatamente, de lo contrario, el robot puede chocar.
- Si se utiliza HTTP en lugar de WebSocket, el robot podría no ser tan sensible debido a la sobrecarga que se ha discutido anteriormente.
- Los WebSockets permiten actualizaciones de datos en tiempo real y la sincronización, el chat de texto en vivo, la videoconferencia, el control y monitoreo de IoT.

¿Cuándo usar WebSockets?

Para determinar si WebSockets son la mejor opción para una tarea, pregúntese:

- ¿Involucra mi aplicación a múltiples usuarios comunicándose entre sí?
- ¿Es una ventana a datos del servidor que cambian constantemente?

Si la respuesta es sí, entonces WebSockets son una excelente opción para mejorar la experiencia de usuario en su aplicación.
Algunas ventajas adicionales de WebSockets incluyen:

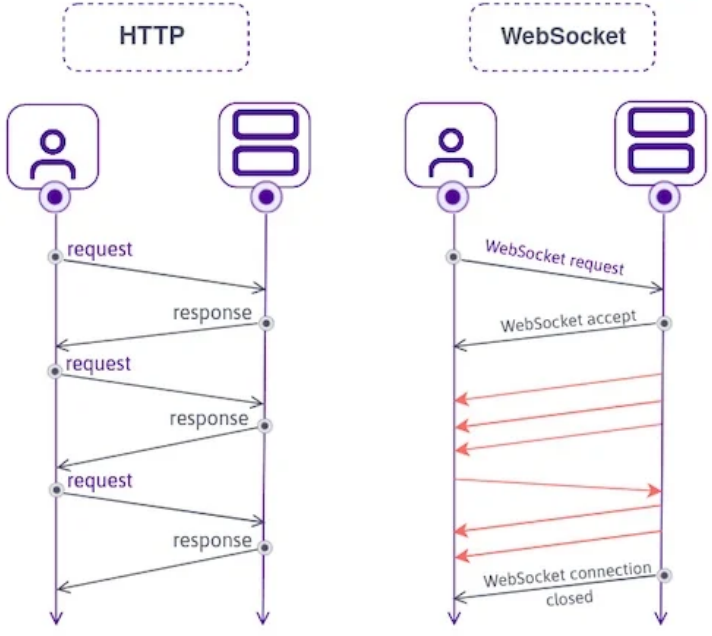
- Mayor velocidad y eficiencia en la transferencia de datos que otros métodos.
- Permite una conexión persistente que permite enviar y recibir datos en cualquier momento sin necesidad de recargar la página.
- Funciona con cualquier lenguaje de programación y cualquier plataforma que soporte TCP.

Ventajas de WebSockets

- Permite una comunicación en tiempo real entre el cliente y el servidor.
- Es más eficiente que otras alternativas como el polling.
- Permite la comunicación bidireccional en tiempo real mediante una única conexión TCP.
- Es la mejor solución para aplicaciones en tiempo real como salas de chat.

Desventajas de WebSockets

- Debido a que WebSockets no es realmente HTTP, algunos de los elementos de infraestructura de TI, como balanceadores de carga o firewalls, podrían no funcionar bien con WebSockets.
- No se recuperan automáticamente las conexiones terminadas. Es necesario implementar esta funcionalidad y por eso existen muchas librerías de lado del cliente.


HTTP
request
response
request
response
request
response
WebSocket
WebSocket request
WebSocket accept
WebSocket connection
closed

Laboratorio de WebSocket

Laboratorio de WebSocket

En este laboratorio, se desarrollará un sistema que permita la lectura de datos de humedad y temperatura de un sensor DHT11 a través de un ESP32. Para ello, se utilizará un código Arduino que permitirá la lectura de los datos y su envío a través de una conexión WebSocket a un servidor en la Raspberry Pi. El servidor, por su parte, contará con un código Python que abrirá el WebSocket para recibir los datos enviados por el cliente ESP32 y mostrarlos en la terminal. El WebSocket se cerrará automáticamente después de 3 minutos o luego de recibir 10 datos. Los objetivos específicos del laboratorio incluyen:

- Desarrollar un código Arduino que permita la lectura de los datos de humedad y temperatura de un sensor DHT11 a través de un ESP32.
- Configurar el ESP32 como cliente para enviar los datos leídos a través de una conexión WebSocket al servidor en la Raspberry Pi cada 5 segundos.
- Desarrollar un código Python que permita al servidor Raspberry Pi abrir el WebSocket y recibir los datos enviados por el cliente ESP32.
- Configurar el servidor para mostrar los datos recibidos en la terminal.
- Configurar el servidor para cerrar automáticamente el WebSocket después de 3 minutos o después de recibir 10 datos.

Laboratorio de WebSocket

Para instalar la librería <WiFi.h> y <WebSocketsClient.h> en Arduino, se deben seguir los siguientes pasos:

1. Abrir el IDE de Arduino.
2. Seleccionar "Sketch.[en] el menú y luego Ïnclude Library".
3. En el menú desplegable, seleccionar "Manage Libraries".
4. En la ventana de administración de bibliotecas, buscar "WiFi.hz "WebSocketsClient.h".
5. Seleccionar ambas bibliotecas y hacer clic en el botón Ïnstall".
6. Esperar a que la instalación se complete y verificar que las bibliotecas aparezcan en la lista de bibliotecas instaladas.
7. En el código de Arduino, agregar las directivas #include <WiFi.h> y #include <WebSocketsClient.h> al principio del archivo.

Laboratorio de WebSocket

Para instalar la librería asyncio y websockets en la Raspberry Pi, se deben seguir los siguientes pasos:

1. Abrir la terminal de la Raspberry Pi.
2. Actualizar los repositorios de paquetes con el comando "sudo apt-get update".
3. Instalar la librería asyncio con el comando "sudo apt-get install python3-asyncio".
4. Instalar la librería websockets con el comando "sudo apt-get install python3-websockets".
5. Esperar a que la instalación se complete y verificar que las librerías se hayan instalado correctamente.
6. Importar las librerías en el código de Python utilizando las directivas ïmport asyncioz ïmport websockets".

¡Muchas gracias por su atención!

¿Preguntas?

Contacto: Marco Teran
webpage: marcoteran.github.io/
e-mail: marco.teran@usa.edu.co

UNIVERSIDAD
SERGIO ARBOLEDA