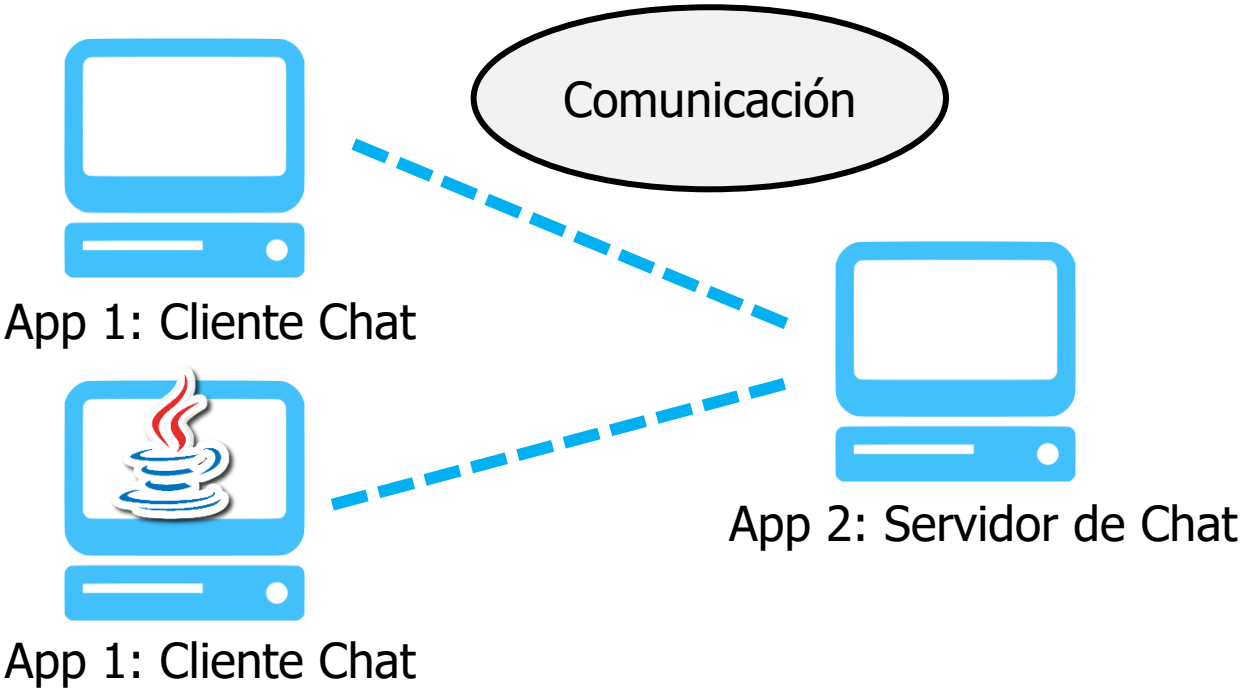
Seminario - Protocolos

Sistema distribuido

- Sus componentes hardware/software están en ordenadores conectados en red.
- Sus acciones se coordinan mediante mensajes para lograr un objetivo
  - Ejemplo (Arquitectura Cliente-Servidor con sockets)



Protocolos de comunicación

- Existen varias clasificaciones de los protocolos
  - Clasificación OSI (Open System InterConnection, interconexión de sistemas abiertos)

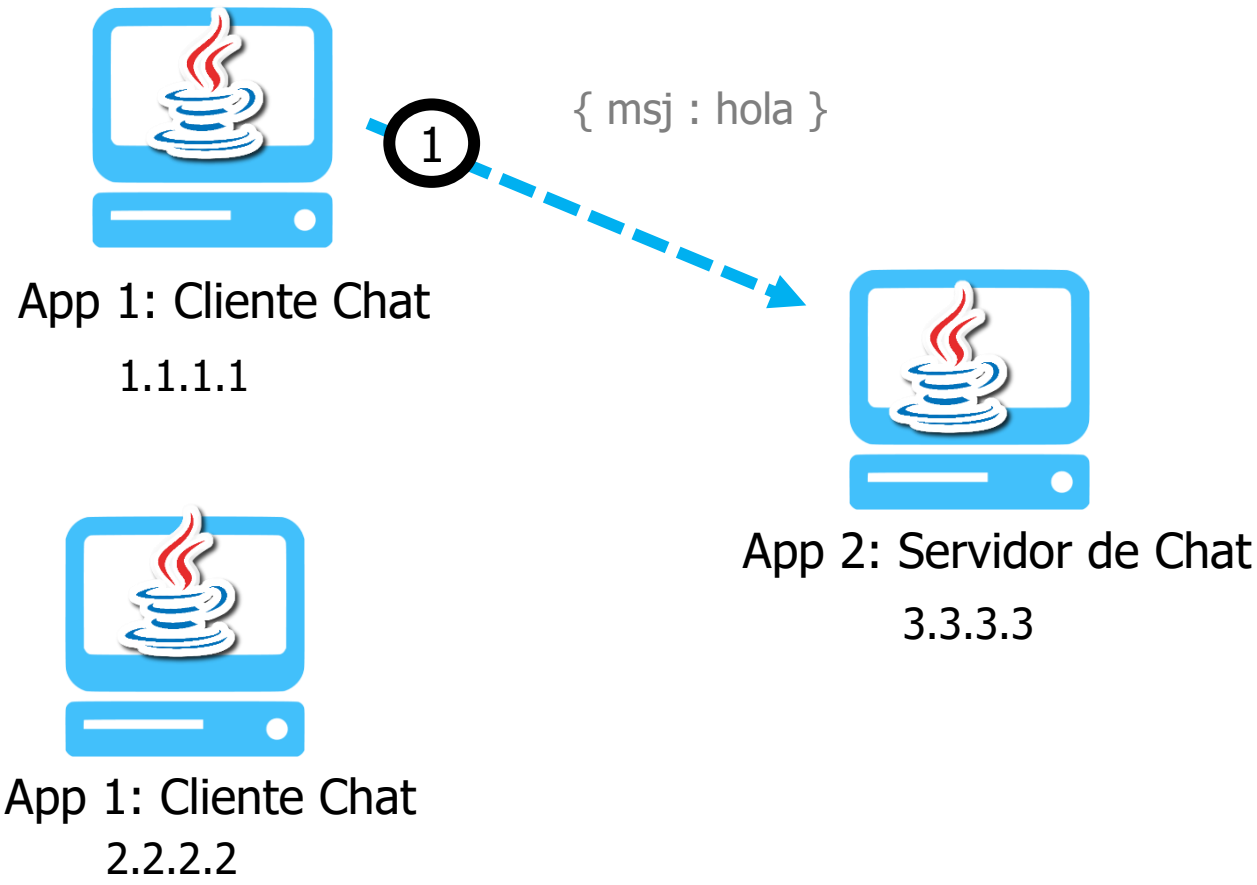
Capa	Nivel	
1	Físico IEEE 802.11x , GSM, Bluetooth, etc.	Transporte de datos
2	Enlace de datos Point-to-point , HDLC, etc.	
3	Capa de Red IP(IPv4, IPv6), OSPF, etc.	
4	**Transporte** **UDP, TCP**	Aplicación
5	Sesión RPC, SCP, ASP	
6	Presentación ASCII, Unicode, EBCDIC.	
7	**Aplicación** **HTTP, HTTPS, FTP, POP, SMTP, SSH, etc.**	

UDP – User/Universal Datagram Protocol

- **No orientado a conexión**
  - Flujo **unidireccional** de una máquina a otra (sin conexión previa)
- El **emisor** envía un paquete de datos al **receptor (debe conocer su IP)**
  - No hay confirmación ni garantía de que el paquete llegue
  - La falta de verificación lo convierte en rápido y ligero
- Útil para transmisión **rápida** de datos y **bajo tráfico de red**
- Utilizado por: DNS, DHCP, TFTP, RIP, VoIp
- Mensajes de 4 campos

UDP – User/Universal Datagram Protocol

- **Comunicación UDP**
  - Unidireccional, se envían a un receptor.



UDP – User/Universal Datagram Protocol

- **Demo UDP Sockets Node.js**
  - Servidor: puede recibir mensajes

```
let datagram = require('dgram');
let servidor = datagram.createSocket('udp4');

servidor.on('listening', function() {
    console.log('Servidor UDP escuchando...');
});

servidor.on('message', function(msjCliente, emisor) {
    console.log(emisor.address + ':' + emisor.port);
    let valorSensor = msjCliente.toString();
    ...
});

servidor.bind(3001, '127.0.0.1');
```

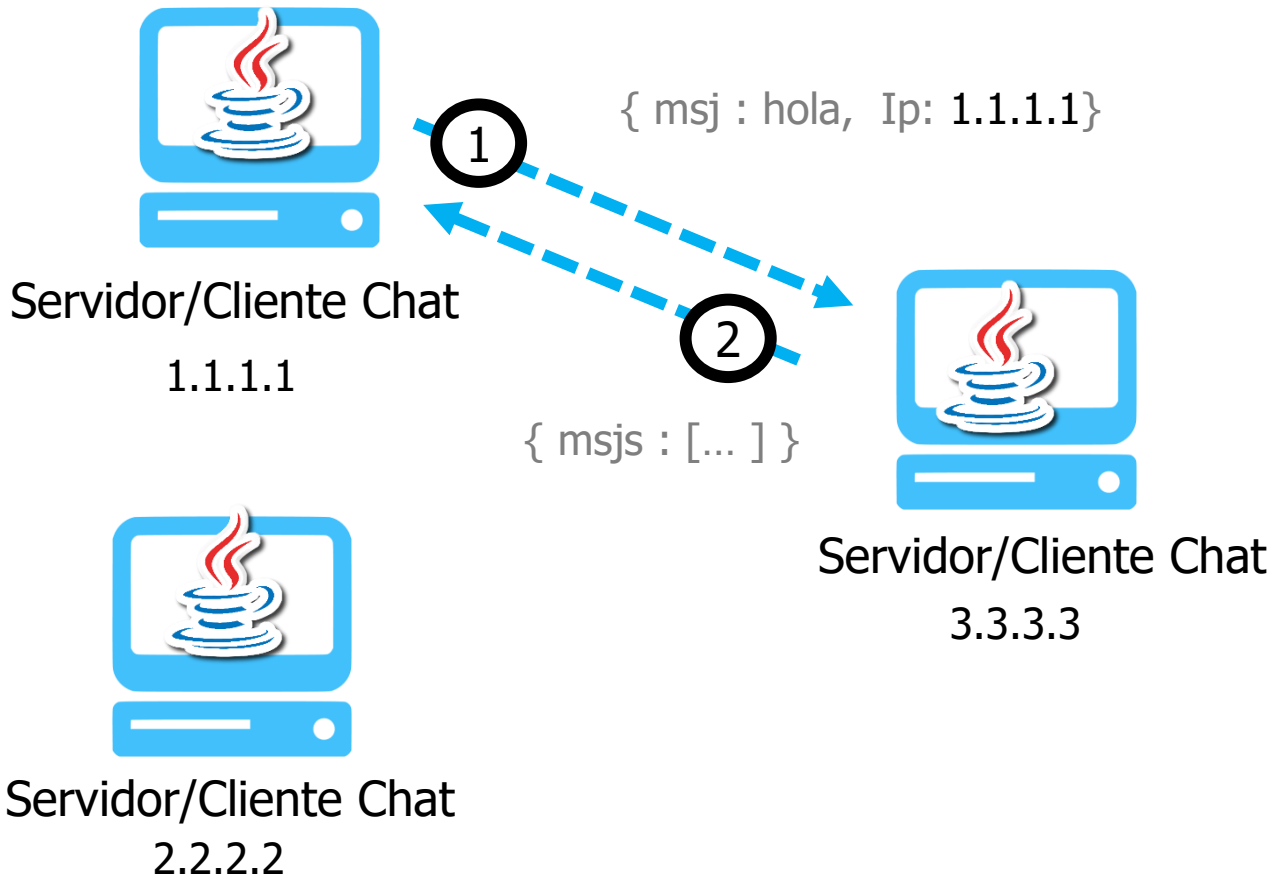
UDP – User/Universal Datagram Protocol

- **Demo UDP Sockets Node.js**
  - Cliente: envía mensaje

```
let datagram = require('dgram');
let cliente = datagram.createSocket('udp4');

var valorSensor = leerSensor();

let msjCliente = new Buffer.from(valorSensor.toString());

cliente.send(msjCliente, 0, msjCliente.length,
                    3001, '127.0.0.1', function(err, bytes) {

    console.log('Mensaje UDP enviado');
    cliente.close();
});
```

UDP – User/Universal Datagram Protocol

- **Comunicación UDP**
  - Todas las aplicaciones podrían ser Clientes y Servidores.

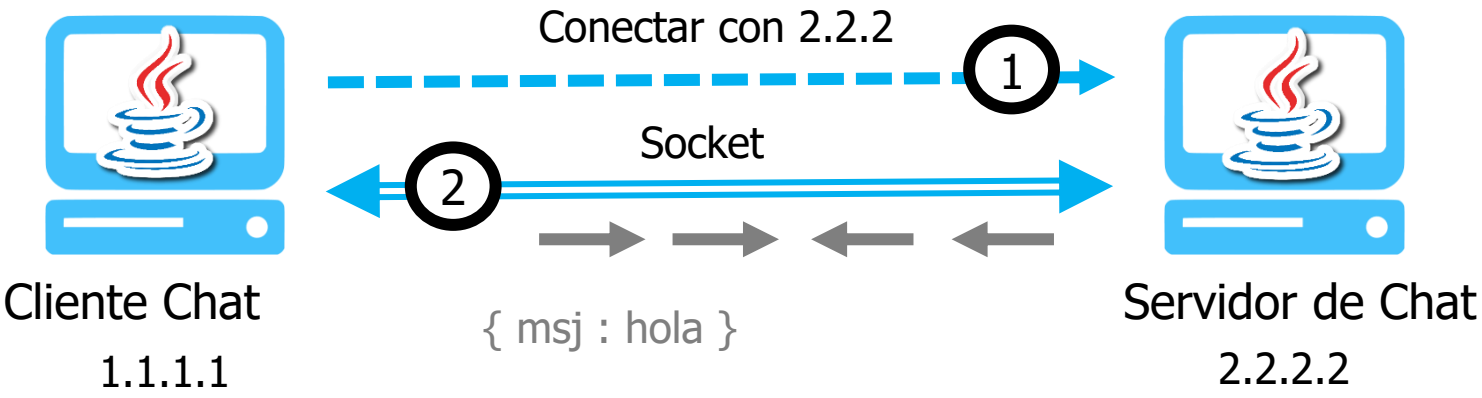


TCP – Transmission Control Protocol

- **Orientado a conexión**
  - El cliente se conecta al servidor
  - Se crea una conexión - canal de comunicación
- El **emisor** y **receptor** pueden intercambiar paquetes por medio de la **conexión**
  - Flujo **bidireccional**
  - Control: seguridad, reenvío de paquetes corruptos…
  - Hay garantía de que los paquetes llegan y lo hacen en orden
- Alta confiabilidad
- Reordena los paquetes en el orden de envió
- Utilizado por : HTTP, HTTPs, SMTP, Telnet
- Mensajes de 12 campos

TCP – Transmission Control Protocol

- **Comunicación TCP**
  - El cliente se **conecta** al servidor
  - Después se crea un **socket de comunicación** bidireccional



App 1: Cliente Chat

TCP – Transmission Control Protocol

- **Demo TCP Sockets Node.js**
  - Servidor – ejecutar **app.js** como aplicación Node normal
  - Cliente 1 y 2 – ejecutar **ejecutar-normal.bat** para abrir aplicación de escritorio

RPC – Remote Procedure Call

- RPC - Llamada a procedimientos remotos
- Ejecuta código/procedimientos de otra máquina **abstrayendo la comunicación**
- Éxito: abstracción sobre los sockets
- Libera al desarrollador de la gestión de la comunicación
- Múltiples implementaciones basadas en RPC:
  - Java RMI – Remote Method Invocation
  - ONC RPC
  - DCE/RPC
  - Otras…

RPC – Remote Procedure Call

- **Demo RPC/TCP Node.js**
  - Servidor: declara procedimientos

```
let rpc = require('node-json-rpc');

let opciones = {port: 5080, host: '127.0.0.1'};
let servidor = new rpc.Server(opciones);

servidor.addMethod('calcular', function(parametros,callback) {
    let error, resultado;
    if (parametros.length == 2)
        resultado=parametros[0] + parametros[1];
    else
        error = { mensaje: 'Solo se aceptan dos parámetros.'};
    callback(error, resultado);
});

servidor.start(function(error) {
    console.log('Servidor RCP iniciado');
});
```

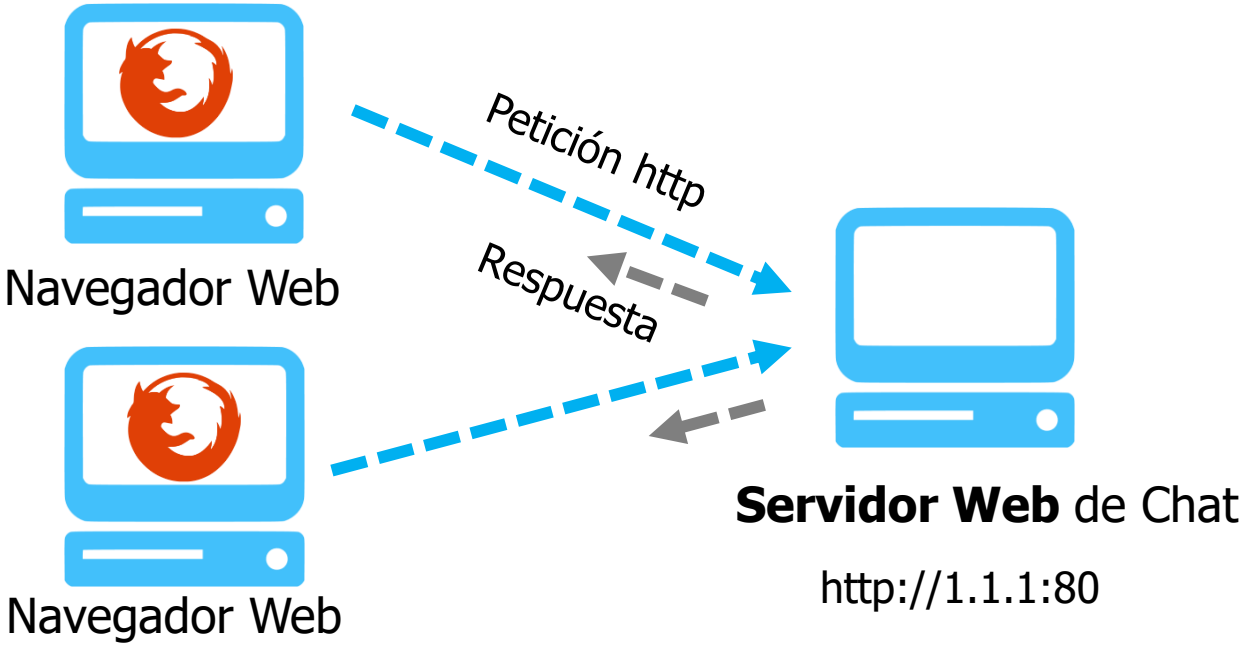
RPC – Remote Procedure Call

- **Demo RPC/TCP Node.js**
  - Cliente: invoca procedimientos

```
let rpc = require('node-json-rpc');

let opciones = {port: 5080, host: '127.0.0.1',};
let cliente = new rpc.Client(opciones);

let a = 20, b = 3;

cliente.call(
    {"method": "calcular", "params": [a,b]},
    function (err, res) {
        if (err)
            console.log("Error:" +err.mensaje);
        else
            console.log('Resultado :' + res.result);
    }
);
```

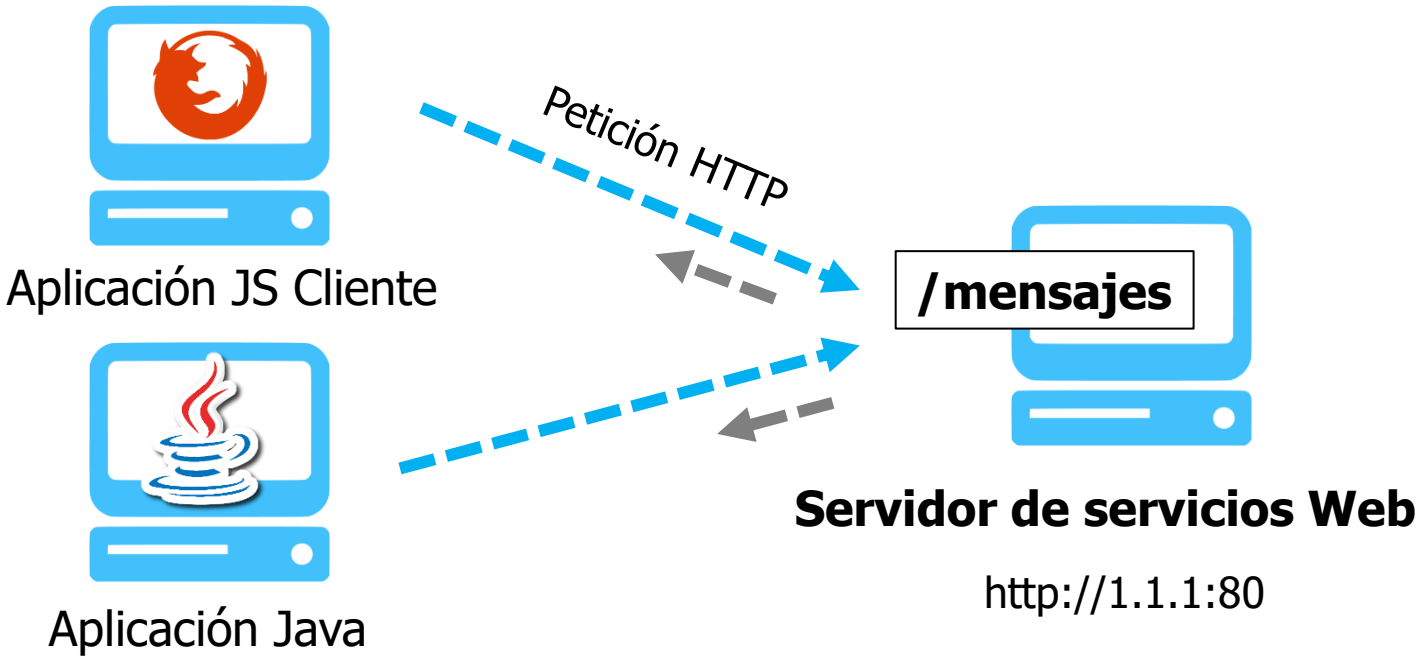
HTTP- Hypertext Transfer Protocol

- Protocolo creado sobre TCP y usado en la Web (y en otras aplicaciones y servicios)
- **Aplicaciones Web:**
  - **Servidor Web** – recibe peticiones HTTP y retorna respuestas basadas en estándares web
  - **Navegador Web -** Realiza peticiones HTTP al servidor e interpreta la respuesta



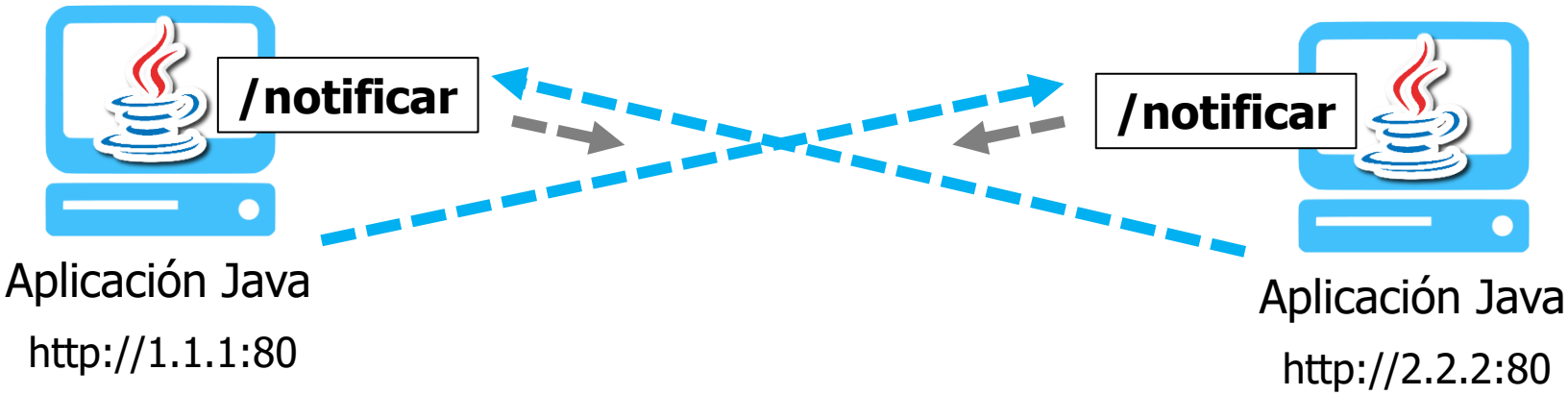
HTTP: Servicios Web

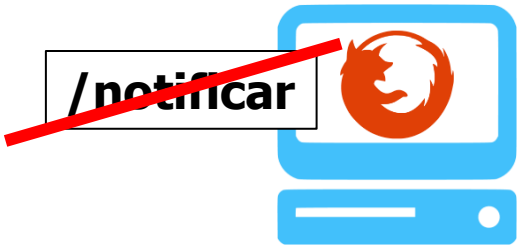
- **Servicios Web**
  - Declara puntos de acceso / URLs a las que se accede mediante HTTP
  - Retornan respuestas en formatos: XML, JSON, etcétera; con el fin de que sean procesadas por las aplicaciones.



HTTP: Servicios Web

- Aplicaciones cliente y servidor (simultáneamente)
  - Cualquiera podría enviar una petición - iniciar una comunicación





No pueden crear un servidor en el **navegador** del usuario

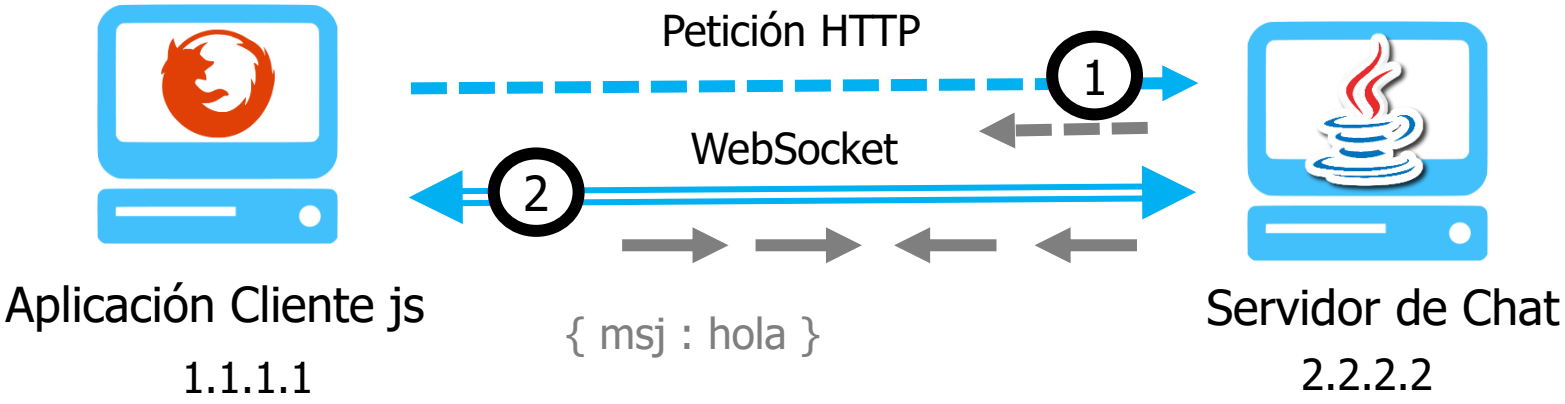
Aplicación JS Cliente

WebSockets

- Permiten abrir una **comunicación** entre **navegador** y **servidor**
  - Envío de mensajes **bidireccional**
- El canal se solicita mediante una **petición HTTP**
  - El **navegador** envía la petición al servidor
  - El servidor responde y se establece la comunicación.
- Posteriormente los mensajes se envían por TCP
  - La comunicación se centra en un único puerto (ejemplos: 80/443)
  - Puede multiplexar diferentes conexiones en un único puerto
- Están implementados en la mayoría de los navegadores

WebSockets

- **Comunicación WebSockets**
  - El cliente / navegador envía una petición HTTP específica al servidor
  - Después se crea un **Web socket de comunicación** bidireccional



Http: WebSockets

- **Demo WebSockets Node.js**
  - Servidor: ejecutar **app.js** como aplicación Node normal
  - Cliente – acceder a http://localhost:8080