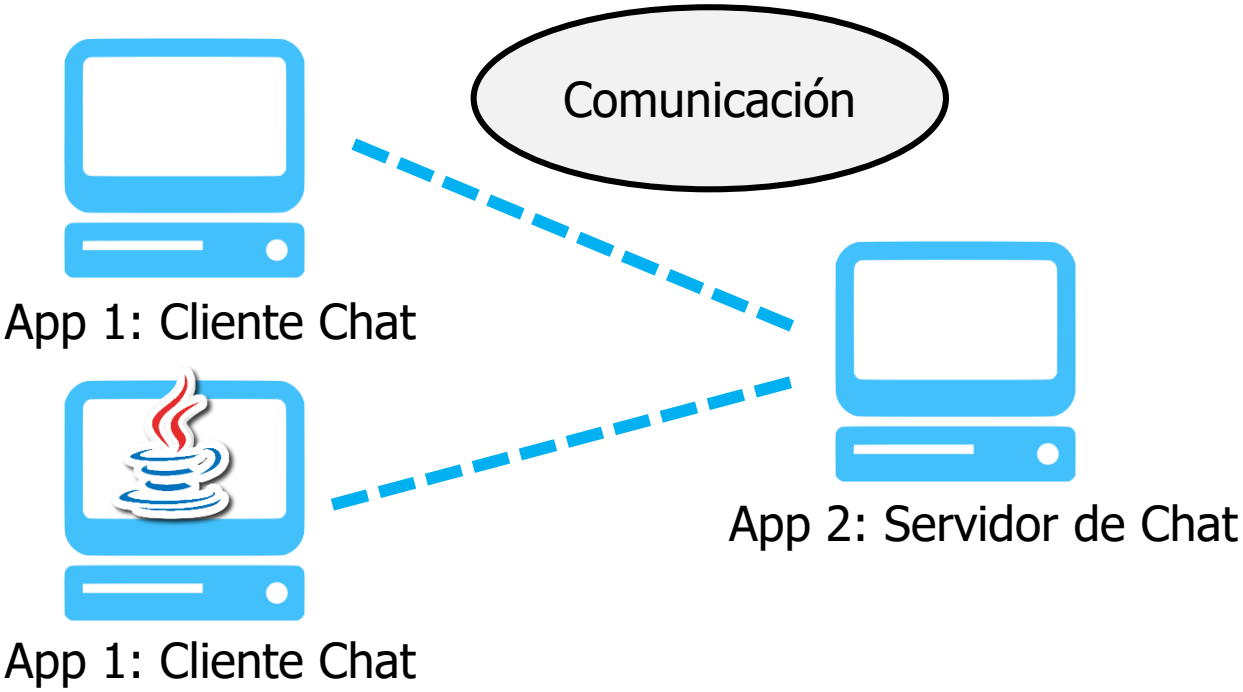
Seminario 7

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)

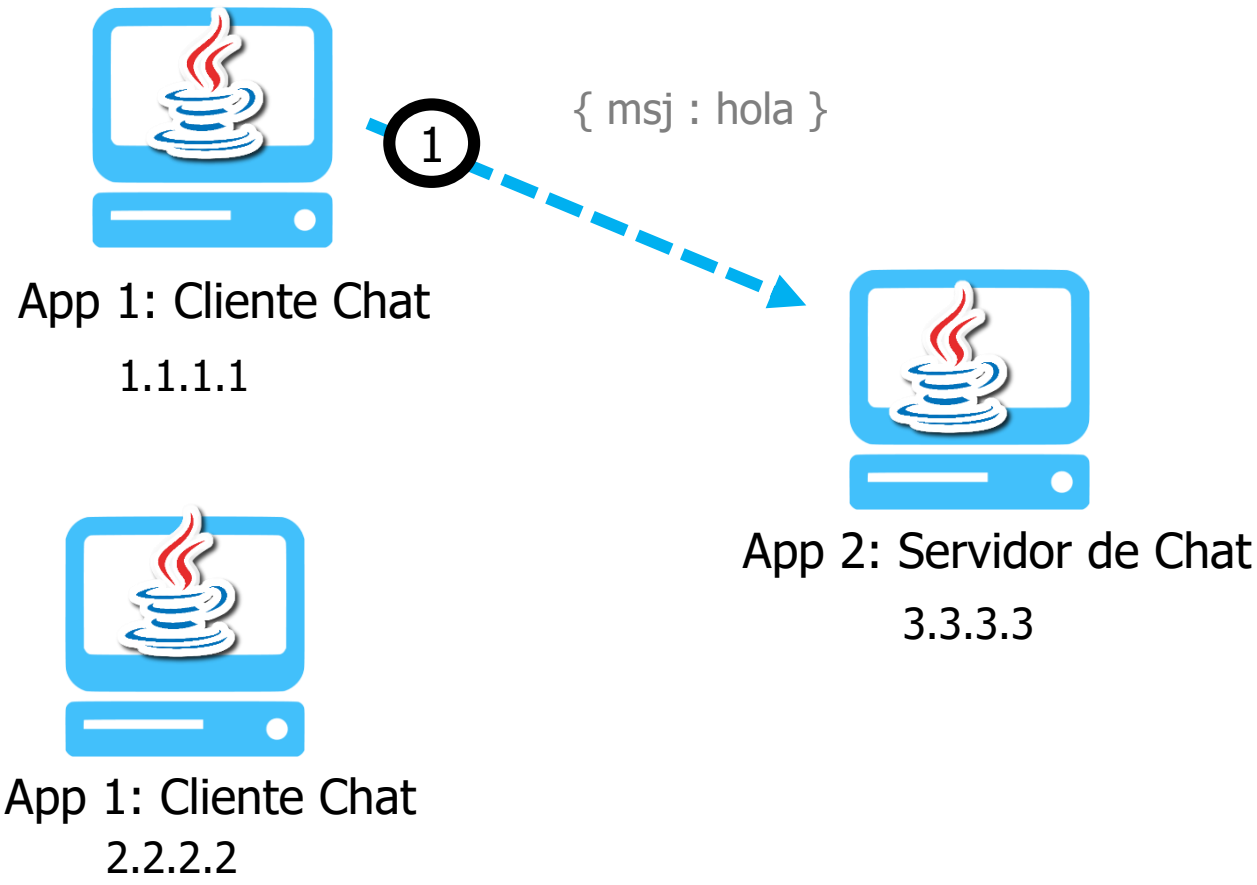
Capa	Nivel	
1	Físico IEEE 802.11x , GSM, Bluetooth, etc.	Transporte de datos
2	Enlace de datos Pont-to-point , HDLC, etc.	
3	Capa de Red IP(IPv4, IPv6), OSPF, etc.	
4	**Transporte** **UDP, TCP**	Aplicación
5	Sesión RPC, SCP, ASP	
6	Presentación ASCII, Unicode, EBCDIC.	
7	**Aplicación** **HTTP, HTTPS,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

```
1  var datagram = require('dgram');
2  var servidor = datagram.createSocket('udp4');
3
4  servidor.on('listening', function () {
5      console.log('Servidor UDP Escuchando ');
6  });
7
8  servidor.on('message', function (mensajeCliente, emisor) {
9      Console.log(emisor.address + ':' + emisor.port);
10     var valorSensor = mensajeCliente.toString();
11     ...
12 });
13
14 servidor.bind(3001, '127.0.0.1');
```

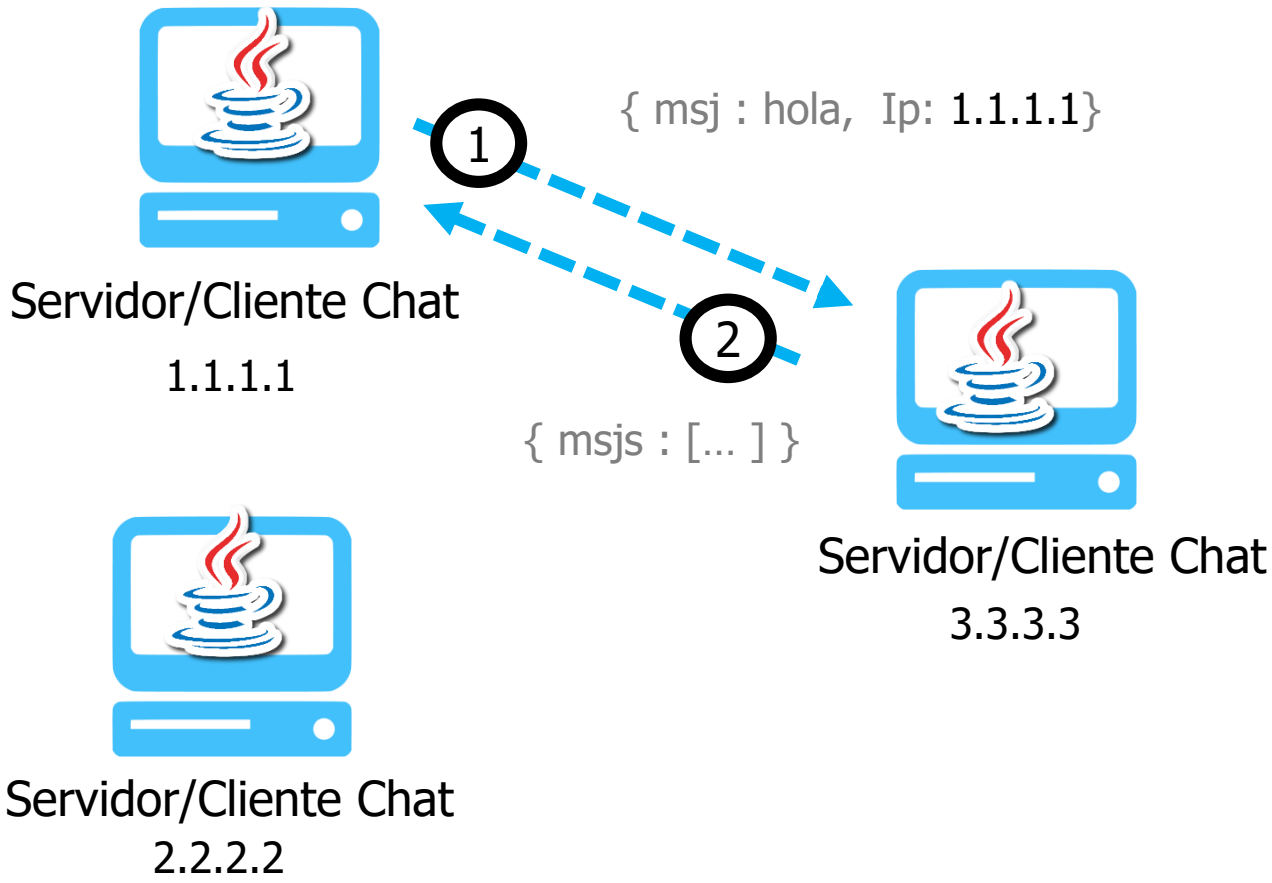
UDP – User/Universal Datagram protocol

- **Demo UDP Sockets Node.js**
  - Cliente – envia mensaje

```
1  var datagram = require('dgram');
2  var cliente = datagram.createSocket('udp4');
3
4  var valorSensor = leerSensor();
5  var msjCliente = new Buffer( valorSensor.toString() );
6
7  cliente.send(msjCliente, 0, msjCliente.length,
8                  3001, '127.0.0.1', function(err, bytes) {
9
10     console.log('Mensaje UDP enviado');
11     cliente.close();
12 });
```

UDP – User/Universal Datagram protocol

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

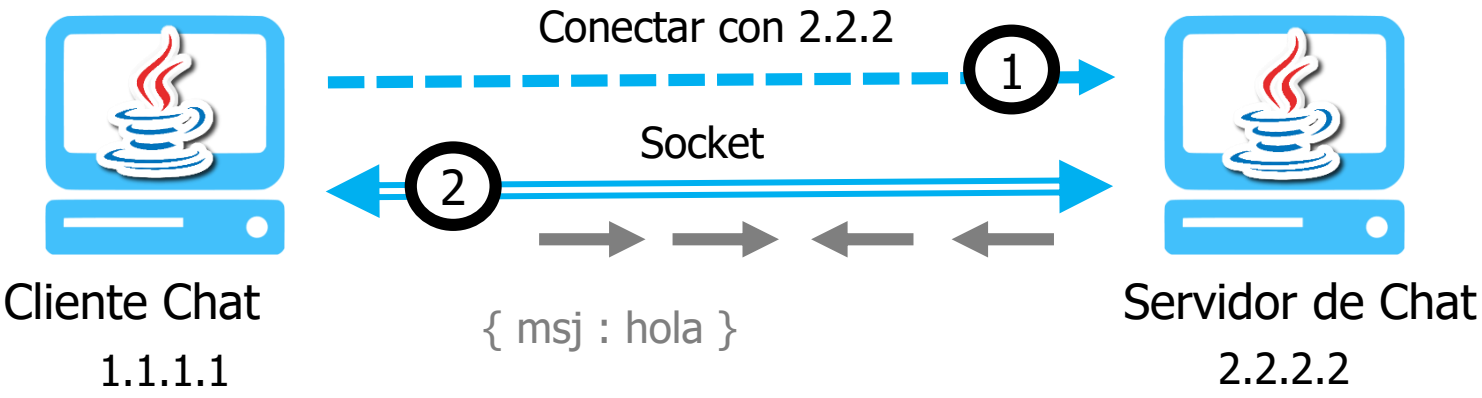


TCP – Transmission Control Protocol

- **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, reenvió de paquetes corruptos…
  - Hay garantía de que los paquetes llegan y lo hacen en el 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: abstraccion 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

```
5  var options = {
6    port: 5080,
7    host: '127.0.0.1'
8  };
9
10 var server = new rpc.Server(options);
11
12 server.addMethod('consultar', function (parametros , callback) {
13   var resultado = parametros[0] + parametros[1];
14   callback(resultado);
15 });
16
17 server.start(function (error) {
18   console.log('Iniciado');
19 });
```

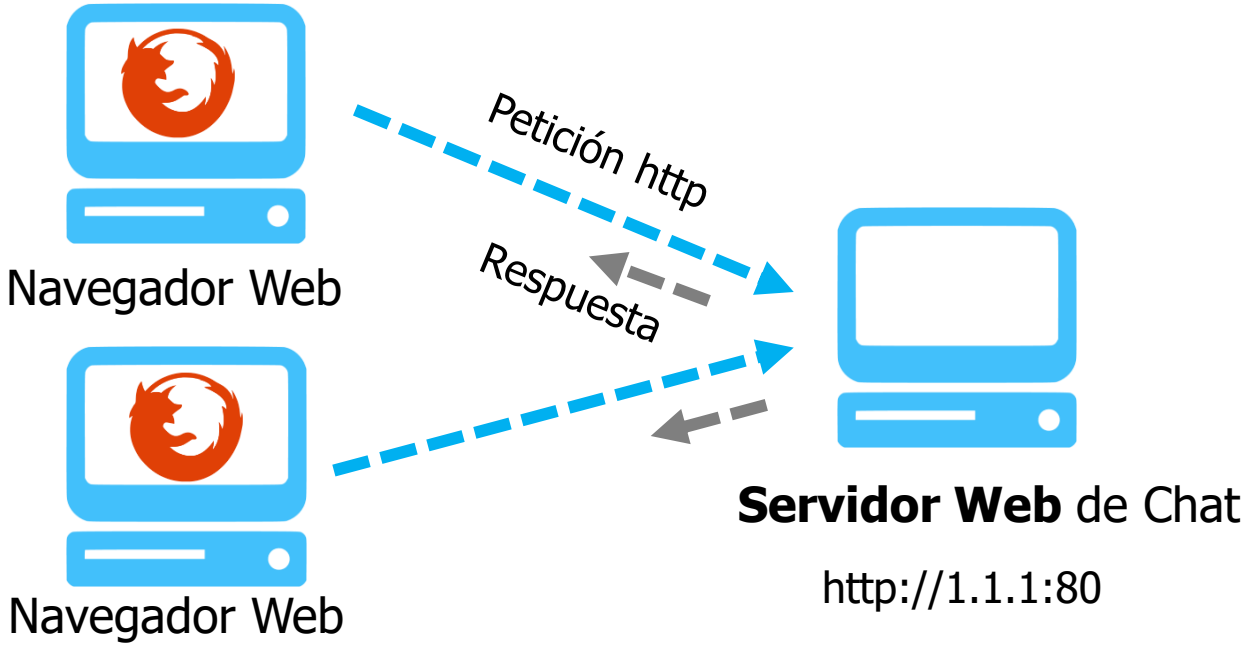
RPC – Remote procedure call

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

```
5  var options = {
6    port: 5080,
7    host: '127.0.0.1',
8  };
9
10 var cliente = new rpc.Client(options);
11
12 cliente.call(
13   {"method": "consultar", "parametros": [1,2]},
14   function (resultado) {
15     console.log(resultado);
16   }
17 );
```

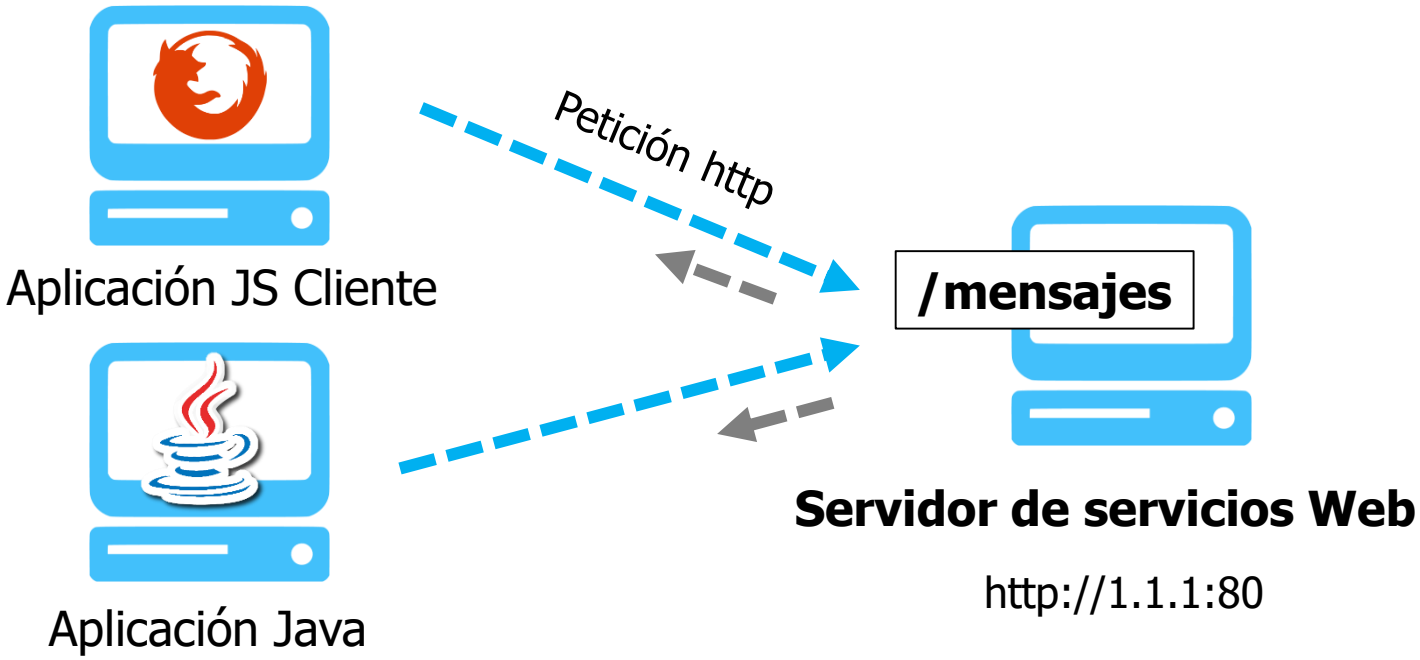
Http - Hypertext Transfer Protocol

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



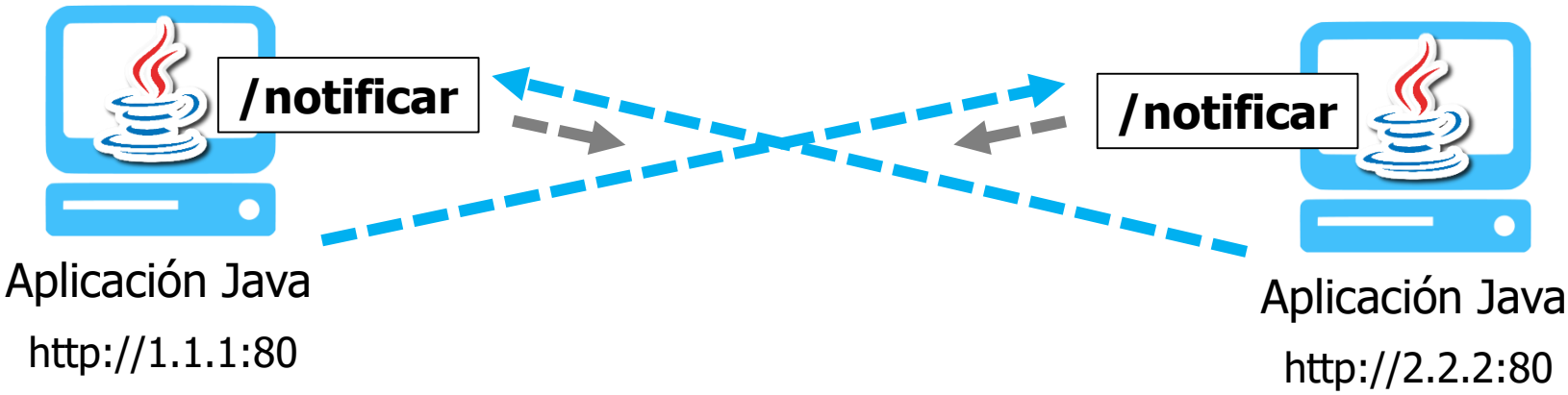
Http: Servicios Web

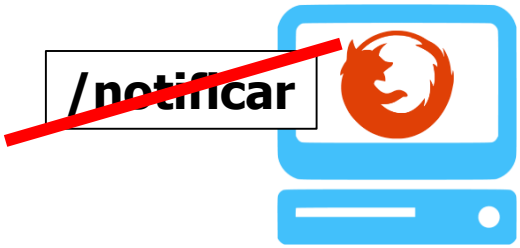
- **Servicios Web**
  - Declara puntos de acceso / URLS a las que se accede con http
  - Retornan respuestas para ser procesadas por aplicaciones (JSON, XML, etc.)



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

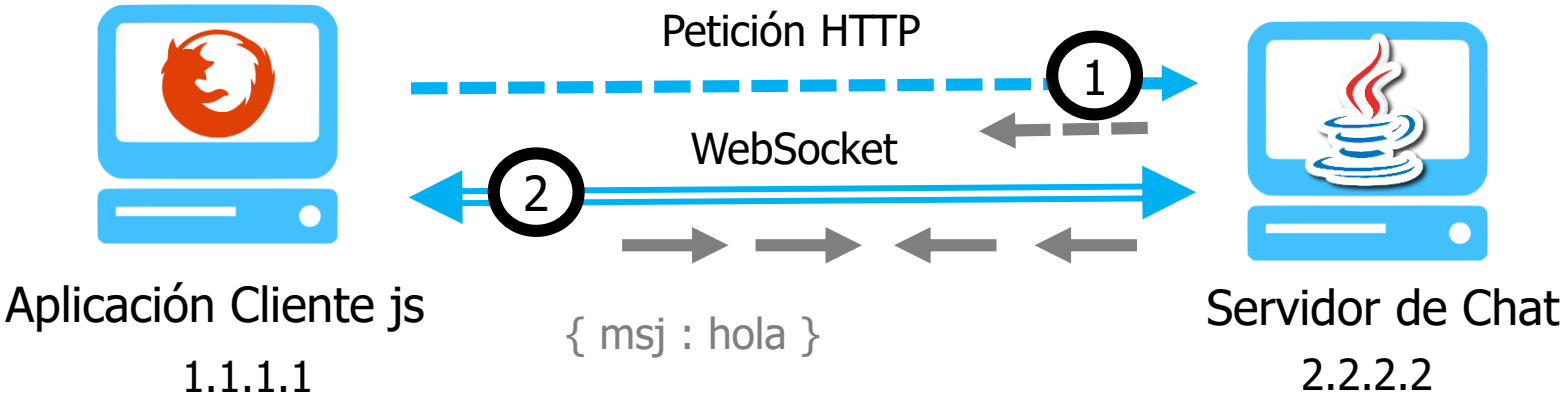
Aplicación JS Cliente

WebSockets

- Permiten abrir una **comunicación** entre **navegador** y **servidor**
  - Envió de mensajes bidireccional
- El canal se solicita con una **petición HTTP** especifica
  - El **navegador** la petición al servidor
  - El servidor responde y se crea la comunicación
- Posteriormente los mensajes se envían por TCP
  - La comunicación se centra en un único puerto ej 80 (libre de Proxies)
  - Puede multiplexar diferentes conexiones en un único puerto
- Están implementados en gran parte de navegadores

WebSockets

- **Comunicación WebSockets**
  - El cliente / navegador envía una petición HTTP especifica 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