HTTP Polling vs WebSockets

A Websocket Application: Weather Station over MQTT

HTTP Polling

HTTP polling is a technique where the client repeatedly requests data from the server at fixed intervals, regardless of whether new data is available. This is inefficient because many requests may return the same data.

Example: Polling Every 5 Seconds

```
function fetchUpdates() {
    var request = new XMLHttpRequest();
    request.open("GET", "server_endpoint.php", true);

    request.onreadystatechange = function () {
        if (request.readyState == 4 && request.status == 200) {
            console.log("Server Response:", request.responseText);
            document.getElementById("data").innerText = request.responseText;
        }
    };

    request.send();
}

// Poll every 5 seconds
setInterval(fetchUpdates, 5000);
```

How it works:

- Every 5 seconds, the browser asks the server for new data.
- The server responds with the latest available data.
- If no new data is available, the request is still sent (wasted bandwidth).

Use Case: Simple applications where real-time updates are not critical, like checking for new emails in a webmail service.

WebSockets

WebSockets provide a persistent connection between the client and server. Unlike polling, the server only sends data when there's an update, making it more efficient.

Example: Real-Time WebSocket Connection

```
const socket = new WebSocket("ws://example.com:9001");

socket.onopen = function () {
    console.log("Connected to WebSocket server.");
    socket.send("Hello Server!");
};

socket.onmessage = function (event) {
    console.log("Received:", event.data);
    document.getElementById("data").innerText = event.data;
};
```

How it works:

- The browser opens a WebSocket connection to the server.
- The server only sends data when there's a real update, reducing unnecessary traffic.
- The connection remains open, so responses are instant.

Use Case: Applications that require real-time updates, like stock market tracking, live chat apps, IoT sensor monitoring, or multiplayer gaming.

Key Differences

- Polling sends frequent requests, even when there's no update.
- WebSockets keep a single connection open and only push updates when necessary.
- Polling wastes bandwidth if the server has no new data, while WebSockets are efficient and deliver instant updates.

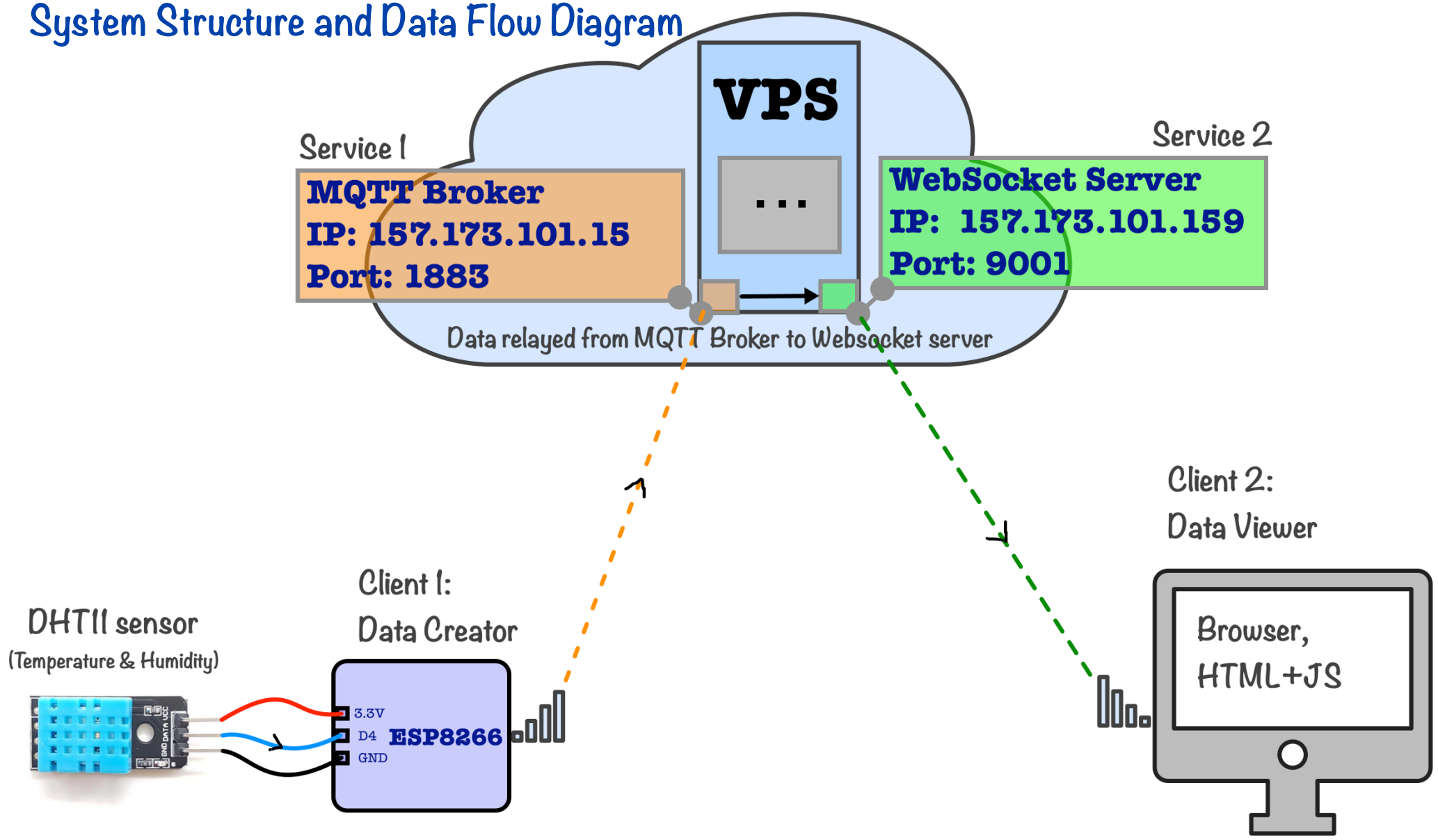
When to Use What?

✓ Use HTTP Polling when updates are infrequent, and real-time data is not critical.
✓ Use WebSockets when low latency and instant updates are required.

Example: If you're building a weather station over MQTT, WebSockets are the best choice since temperature and humidity updates can happen unpredictably.

Weather Station over MQTT

System Structure and Data Flow Diagram



Structure

🌍 VPS (Server)

Service 1: MQTT Broker → 157.173.101.159:1883
Handles sensor data publishing & subscribing.

Service 2: WebSockets Server → 157.173.101.159:9001
• Enables real-time updates in the browser.

Client 1 (Data Creator): Device: ESP8266 + DHT11; Role: Reads temperature & humidity, Publishes to MQTT (1883).

Client 2 (Data Viewer): Device: PC Browser (HTML + WebSockets);
Role: Subscribes to MQTT WebSockets (9001) to display real-time data.

Data Flow Diagram (DFD)

1️⃣ ESP8266 Reads DHT11 sensor values (Temperature & Humidity).

2️⃣ ESP8266 Publishes the readings to MQTT Broker (1883) under topics:
/weather/temp
/weather/humidity

3️⃣ MQTT Broker Relays Data to WebSockets Server (9001).

4️⃣ WebSockets Server Pushes Data to Web Interface (HTML + JavaScript).

5️⃣ Web Interface Displays Live Updates on the browser.