🌐HTTP vs WebSockets — One-Page Revision Notes

1️⃣HTTP Basics

🔹 HTTP/1.0

- Client sends request → Server sends response
- Connection closes after response
- Strict **request–response** model
- ❌No persistent connection
- ❌WebSockets not possible

```
Client   ──Request──▶   Server
Client   ◀─Response──   Server
(Connection closed)
```

🔹 HTTP/1.1

- Supports **persistent connections (keep-alive)**
- Still **client-initiated only**
- Server cannot push data on its own
- ✅Required for WebSocket handshake

```
Client   ──Request──▶   Server
Client   ◀─Response──   Server
(Connection kept alive, but server waits)
```

2️⃣WebSockets (High Level)

- WebSocket is a **separate protocol**, not HTTP
- Uses **HTTP/1.1 only for handshake**
- After upgrade → switches to **WebSocket protocol**
- Enables **real-time, full-duplex communication**

☸️Key idea:

> HTTP is used as a *vehicle* to establish WebSocket connection

3️⃣WebSocket Handshake (Very Important)

🔹 Steps

1. Client connects using:
2. `ws://` (non-secure)
3. `wss://` (secure over TLS)
4. Client sends **HTTP/1.1 request** with headers:
5. `Upgrade: websocket`
6. `Connection: Upgrade`
7. Server responds with:
8. `HTTP/1.1 101 Switching Protocols`
9. Protocol switches from **HTTP → WebSocket**
10. Real-time communication starts

```
Client ──HTTP/1.1 + Upgrade──▶ Server
Client ◀──101 Switching────── Server
        (Protocol switched)
Client ⇄⇄⇄⇄⇄⇄⇄⇄⇄⇄ Server
      (WebSocket frames)
```

☸️ `101` status code is **mandatory**

4️⃣After Handshake (How WS Works)

- Connection stays **open**
- Client & server can send messages **anytime**
- Uses lightweight **frames**, not HTTP messages
- Much lower latency than HTTP

```
Client  ⇄  Server
(real-time, full-duplex)
```

5️⃣WebSocket Use Cases

- 💬Chat applications (WhatsApp, Slack)
- 🆚Live feeds (stocks, crypto, sports)

- 🆔Multiplayer games
- 🈂️Progress updates / live logs
- Collaborative tools (editors, whiteboards)

6️⃣WebSocket Examples

- Browser ↔ Node.js server
- Angular/React ↔ Backend
- Real-time notifications
- IoT live telemetry

7️⃣WebSocket Pros (Advantages)

✅Full-Duplex

- Client & server send independently

✅Low Latency

- No repeated HTTP headers
- Faster data transfer

✅Server Push

- Server can push data without client request

✅HTTP Compatible

- Works over ports **80 / 443**

✅Firewall Friendly

- Usually allowed through corporate firewalls

8️⃣WebSocket Cons (Disadvantages)

❌Proxying is Tricky

- Some proxies don’t support upgrade well

❌Load Balancing (L7)

- Long-lived connections
- Requires sticky sessions / affinity

❌Stateful Connections

- Server keeps client state

- Harder to scale horizontally
- Often needs Redis / Pub-Sub

❌Resource Intensive

- Each open connection uses memory

❌Overkill for Simple Apps

- REST/HTTP better for CRUD

9️⃣HTTP vs WebSocket (Quick Compare)

Feature	HTTP	WebSocket
Communication	Request-Response	Full-Duplex
Server Push	❌	✅
Connection	Short-lived	Long-lived
Latency	Higher	Low
Scaling	Easier	Harder
Use Case	APIs, CRUD	Real-time

⚫Final Takeaway

HTTP is stateless and client-driven.
WebSockets are stateful and event-driven.
WebSockets trade scaling simplicity for real-time performance.

☸️*Perfect for interviews, quick revision, and real-time system design basics.*