Interview Questions - Real time communication

1️⃣ What is real-time communication, and why is it important?

🔹 **Answer:**
Real-time communication (RTC) refers to **instantaneous data exchange** between systems with minimal latency. Unlike traditional request-response models where data retrieval happens on demand, RTC ensures continuous, live updates without requiring users to manually refresh or trigger requests.

🔹 **Why is it important?**

- **Low Latency:** Enables immediate transmission of information.
- **Seamless User Experience:** No need for manual refreshes or delays.
- **Essential for Critical Applications:** Used in **chat applications, live streaming, stock market updates, online gaming, and IoT systems** where timely data delivery is crucial.

🔹 **Examples:**

- WhatsApp messaging
- Live stock market tickers (NASDAQ, NYSE)
- Multiplayer gaming (Fortnite, Call of Duty)
- Live sports score updates

2️⃣ How does WebSockets work, and how does it differ from traditional HTTP?

🔹 **Answer:**
WebSockets provide a **persistent, full-duplex connection** over a single TCP connection, allowing both the client and server to send data at any time.

🔹 **How WebSockets Work:**
1️⃣ The client sends an **HTTP upgrade request** to the server.
2️⃣ If the server supports WebSockets, it responds with a **101 Switching Protocols** status.
3️⃣ A **persistent connection** is established, eliminating the need for repeated HTTP

requests.
4️⃣ Both client and server can exchange messages **asynchronously** without re-establishing the connection.

🔹 **Differences from Traditional HTTP:**

Feature	WebSockets	Traditional HTTP
Connection	Persistent, full-duplex	Connection closes after each request-response cycle
Latency	Low (data sent instantly)	Higher (requires a new request each time)
Communication	Bi-directional (both client and server can send data)	Client-initiated only
Overhead	Minimal (uses a single connection)	High (repeated requests create overhead)

🔹 **Example:**

- **WebSockets:** Used in live chat apps where messages are pushed in real-time.
- **Traditional HTTP:** Used in blogs or static web pages where data doesn't change frequently.

3️⃣ Explain the WebSocket handshake process.

🔹 **Answer:**
The WebSocket handshake is the **initial request-response process** that upgrades an HTTP connection to a WebSocket connection.

🔹 **Steps in the WebSocket Handshake:**
1️⃣ The **client sends an HTTP request** with an `Upgrade` header requesting a WebSocket connection:

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9YZrd6w==
Sec-WebSocket-Version: 13
```

2️⃣ The **server responds** with an HTTP 101 Switching Protocols status if it supports WebSockets:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

3️⃣ After the handshake, the **WebSocket connection remains open**, allowing both the client and server to exchange messages in real-time.

🔹 **Why is the WebSocket Key Used?**

- The `Sec-WebSocket-Key` ensures that the request is not mistakenly interpreted as a standard HTTP request.
- The server hashes this key and sends back the response in `Sec-WebSocket-Accept`.

4️⃣ What is long polling, and how does it work?

🔹 **Answer:**
Long polling is a technique where the **client sends an HTTP request to the server and waits for a response until new data is available**. If there is no new data, the server holds the request open instead of responding immediately.

🔹 **How It Works:**
1️⃣ The **client sends an HTTP request** to the server.
2️⃣ The **server does not immediately respond**. Instead, it holds the request open until there is new data.
3️⃣ Once new data is available, the **server responds**.
4️⃣ The **client immediately sends a new request** to listen for further updates.

🔹 **Example:**

- Gmail uses **long polling** to check for new emails without refreshing the page.

🔹 **Key Differences from WebSockets:**

Feature	WebSockets	Long Polling
Connection	Persistent	Multiple HTTP requests

Efficiency	Low overhead	High overhead due to repeated requests
Latency	Lower	Higher
Best For	High-frequency updates	Less frequent updates

🔹 INTERMEDIATE QUESTIONS

5️⃣ What are the advantages of WebSockets over long polling?

🔹 **Answer:**
✅ **Persistent Connection:** WebSockets keep a **single connection** open, reducing overhead.
✅ **Low Latency:** Data is **pushed in real-time**, unlike long polling where the client waits.
✅ **Efficient:** Avoids unnecessary HTTP headers, reducing **network traffic**.
✅ **Bidirectional Communication:** Both client and server can **send messages** anytime.

🔹 **When WebSockets are better:**

- **Live chat applications** (Slack, WhatsApp)
- **Stock market updates** (NASDAQ)
- **Multiplayer gaming** (Fortnite)

6️⃣ In what scenarios would you prefer long polling over WebSockets?

🔹 **Answer:**
Use **long polling** when:
✅ **WebSockets are not supported** (e.g., older browsers or firewalls blocking WebSockets).
✅ **Limited real-time needs** (e.g., notifications don't require instant updates).
✅ **RESTful API integration** (long polling works well with standard HTTP).
✅ **Simpler backend infrastructure** (no need for a WebSocket server).

🔹 **Example Use Cases:**

- Social media feed updates (Facebook, Twitter)
- Email notifications (Gmail new mail alerts)

7️⃣ How does WebSockets handle connection failures or network interruptions?

🔹 **Answer:**
1️⃣ **Automatic reconnection:** Most WebSocket clients implement **reconnection logic** in case of failure.
2️⃣ **Heartbeat mechanism:** WebSockets use **ping/pong messages** to detect connectivity issues.
3️⃣ **Backoff strategies:** Clients retry connections with **increasing time intervals** to avoid excessive requests.

🔹 **Example Implementation:**

- **Slack WebSockets:** If a user switches between Wi-Fi and mobile data, Slack **automatically reconnects** without losing messages.

8️⃣ Can you use WebSockets with load balancers? If yes, how?

🔹 **Answer:**
Yes, WebSockets can be used with load balancers, but they require **sticky sessions** or **connection-aware load balancing** to maintain state.

🔹 **Techniques:**
✅ **Sticky Sessions:** Ensures that a WebSocket connection stays on the same server.
✅ **Reverse Proxy (NGINX/HAProxy):** Supports WebSocket proxying by **passing the `Upgrade` header**.
✅ **WebSocket Gateways:** Tools like **AWS API Gateway** manage WebSocket connections at scale.

🔹 **Example:**

- Slack uses **AWS Elastic Load Balancer (ELB)** with sticky sessions for WebSockets.

9️⃣ What are some challenges of scaling WebSockets in a distributed system?

🔹 **Answer:**
⚠️ **Maintaining state across multiple servers** (requires sticky sessions or session persistence).
⚠️ **Handling large concurrent connections** (millions of users).

⚠ **Load balancing issues** (WebSockets require intelligent routing).
⚠ **Handling connection failures** (requires reconnection logic).

🔹 **Solution:**

- Use **Redis Pub/Sub** or **Kafka** for **message broadcasting across multiple WebSocket servers**.
- Implement **sticky sessions** in load balancers.
- Use **WebSocket gateways** like AWS API Gateway for scalability.