Going Live: Real-Time Magic in Oracle APEX

Vito Van Hecke – Kevin Thyssen
March 26 – Ede, The Netherlands

Extend APEX like a PRO

APEX Office Print
Printing & Reporting
built for APEX

APEX Media Extension
Image processing for the
Oracle Database.

APEX Office Edit
Document management
for APEX

Plug-ins Pro
Highly requested features
and functionality to
extend APEX

APEX Message Service
Real-time communication
for Oracle APEX

APEX Project Eye
Productivity, Quality Assurance,
and Security tool for APEX

Oracle ACE Associate

Senior APEX consultant @ United Codes

Located in Belgium

15+ years experience in Oracle APEX

Cycling enthousiast

@kthyssen.bsky.social

@kevinthyssen

kevin.thyssen@united-codes.com

kevinthyssen

APEX consultant & Product developer @ United Codes

Located in Belgium

5+ years experience in Oracle APEX

@vvanhecke.bsky.social

@vvanhecke2000

vito.vanhecke@united-codes.com

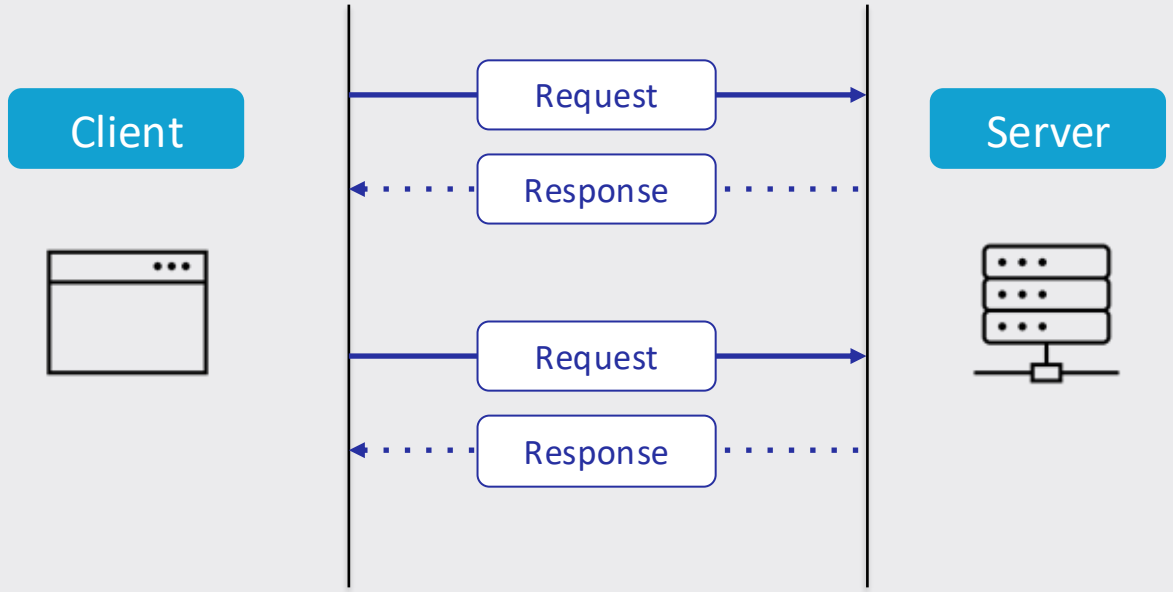
vito-vanhecke

Why real-time communication?

- Traditional HTTP request/response is insufficient for live updates
- Modern apps need instant data flow (e.g., chat, stock tickers, games, video calls, ...)
- Four main techniques:
  - **(Long) Polling**
  - **Server-Sent Events (SSE)**
  - **WebSockets**
  - **WebRTC**
  - WebTransport (relatively new, not widely supported yet)

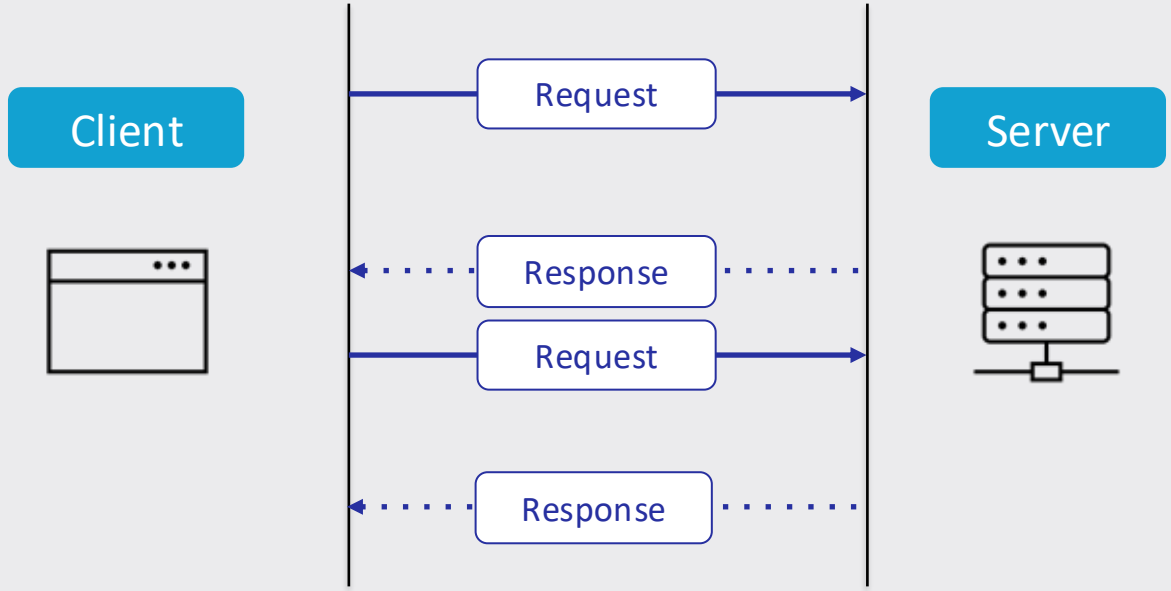
Polling

- Client sends HTTP requests to the server
- Server responds with data, or with an empty message
- After a predetermined time interval, the client sends a new request

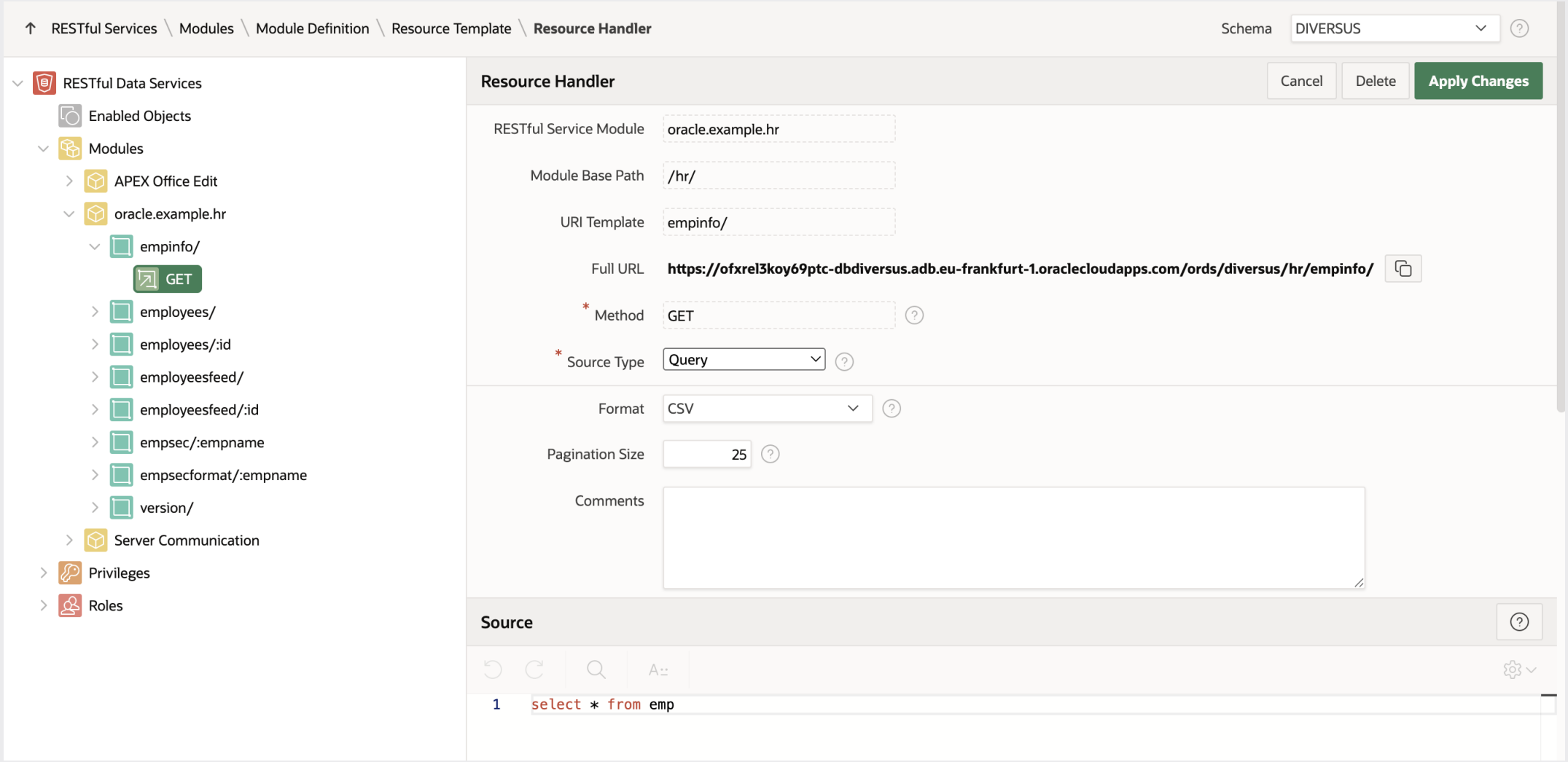


Long Polling

- Client sends HTTP requests to the server
- Instead of responding immediately with empty data, the server keeps the connection open if there are no updates available
- Once the server has new data, it sends the data to the client and closes the connection
- The client immediately makes another request, starting the process again



(Long) Polling – Server Side



Polling – Client Side

```
function poll() {
    fetch('http://127.0.0.1/poll’)
        .then(response => response.json())
        .then(data => {
            console.log("Received data:", data);
        })
        .catch(error => {
            console.error("Error during polling:", error);
        });
}

setTimeout(poll, 5000);
```

Long Polling – Client Side

```
function longPoll() {
    fetch('http://127.0.0.1/poll’)
        .then(response => response.json())
        .then(data => {
            console.log("Received data:", data);
            longPoll();
    })
    .catch(error => {
        setTimeout(longPoll, 10000);
    });
}


longPoll();
```

(Long) Polling

- Easy to implement with plain HTTP
- Works everywhere (no special protocol support needed)
- Good fallback when WebSockets/SSE aren’t available
- Only technology natively supported in Oracle APEX

- High overhead (many HTTP requests)
- Latency between requests (not truly real-time)
- Doesn’t scale well under heavy load

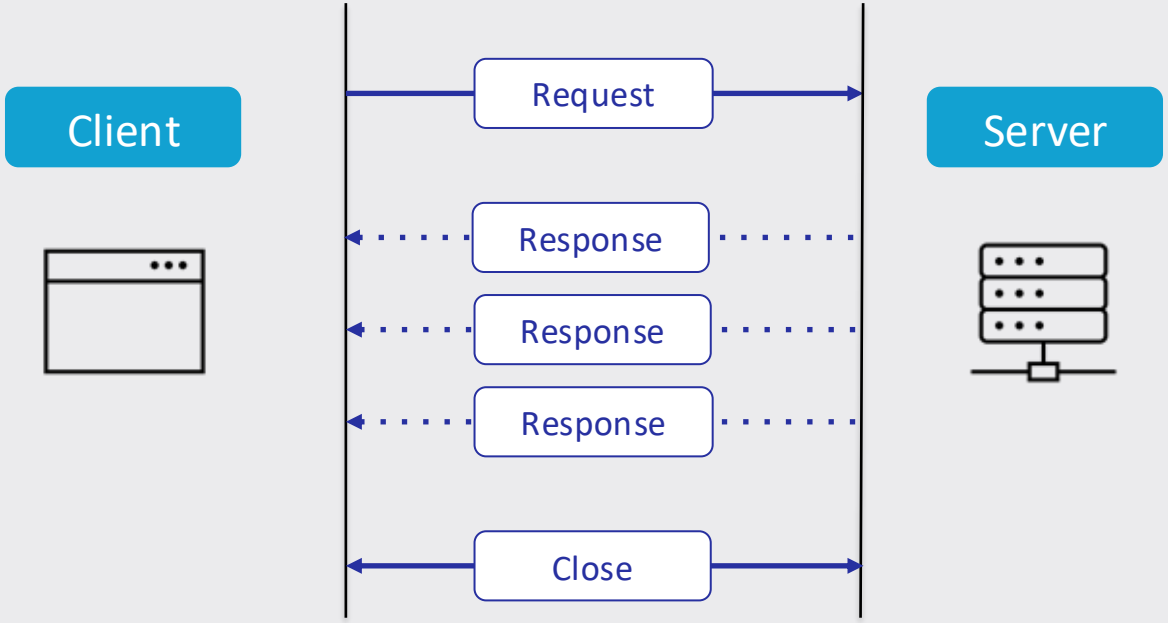
Use cases: Legacy apps, simple chat

Demo

 External server required 

Server-Sent Events (SSE)

- One-way communication: server → client only
- Client subscribes with an HTTP request
- Server streams updates as events
- ⚠ Type: eventsource



SSE – Server Side

```
var SSE = require('express-sse');
var sse = new SSE();
app.get('/stream', sse.init);


let content = 'Test data at ' + JSON.stringify(Date.now());
sse.send(content);
```

SSE – Client Side

```
const evtSource = new EventSource("/stream");


eventSource.addEventListener('message', event => {
   console.log(event.data);
}
```

SSE

- Simple API (EventSource in browsers)
- Lightweight compared to WebSockets
- Automatic reconnection built-in
- Great for streaming data from server to many clients

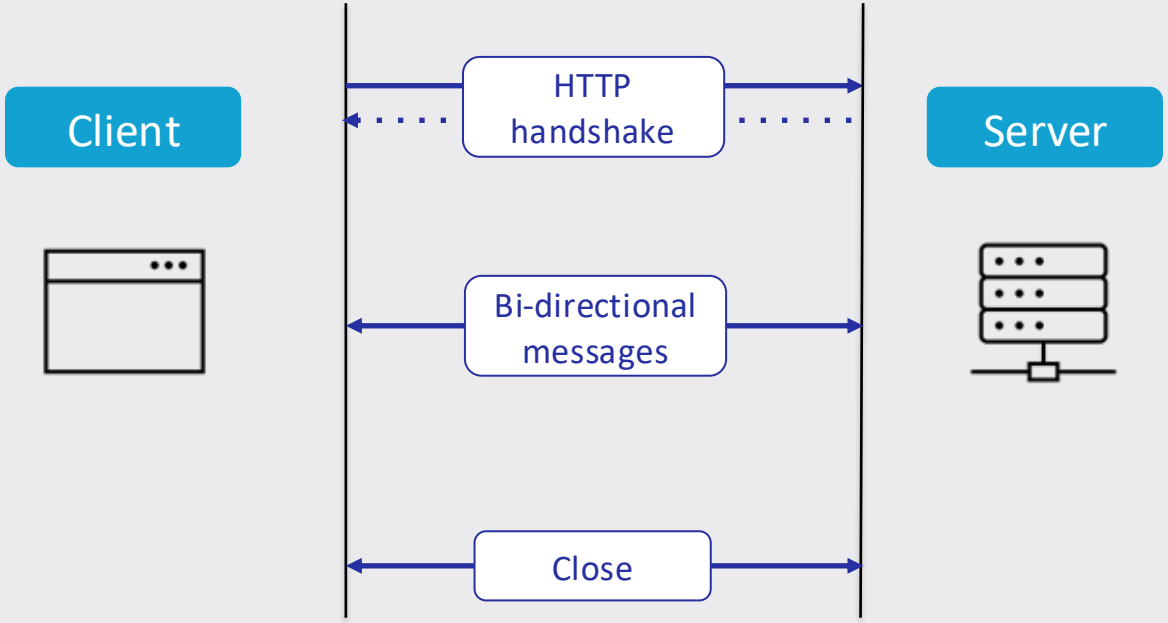
- One-way only (server → client)
- No binary data
- Limit of 6 concurrent connections per browser (HTTP 1.1)
- Limited support in older browsers

Use cases: Live news feed, stock ticker, notifications

Demo

WebSockets

- Full-duplex (two-way) communication over a single TCP connection
- Starts as an HTTP handshake, then upgrades to WebSocket protocol
- Allows real-time, low-latency communication between client and server



WebSockets – Server Side

```
const WebSocket = require('ws');

const wss = new WebSocket.Server({ port: 8080 });

wss.on('connection', function connection(ws) {
   ws.on('message', function incoming(message) {
      console.log('received: %s', message);
   });

   ws.send('something');
});
```

WebSockets – Client Side

```
const ws = new WebSocket(‘ws://127.0.0.1');

ws.addEventListener('open', () => {
    ws.send('Hello!');
});

ws.addEventListener('message', event => {
    console.log('Received:', event.data);
});
```

WebSockets

- Full-duplex (bi-directional) communication
- Low latency, efficient once connected
- Binary support
- Scales better than long polling for real-time needs

- More complex to implement/debug than HTTP
- Not ideal for one-way streams (overkill vs. SSE)
- Complex to scale over multiple servers / load balancers

Use cases: Chat apps, multiplayer games, collaborative tools

Socket.IO

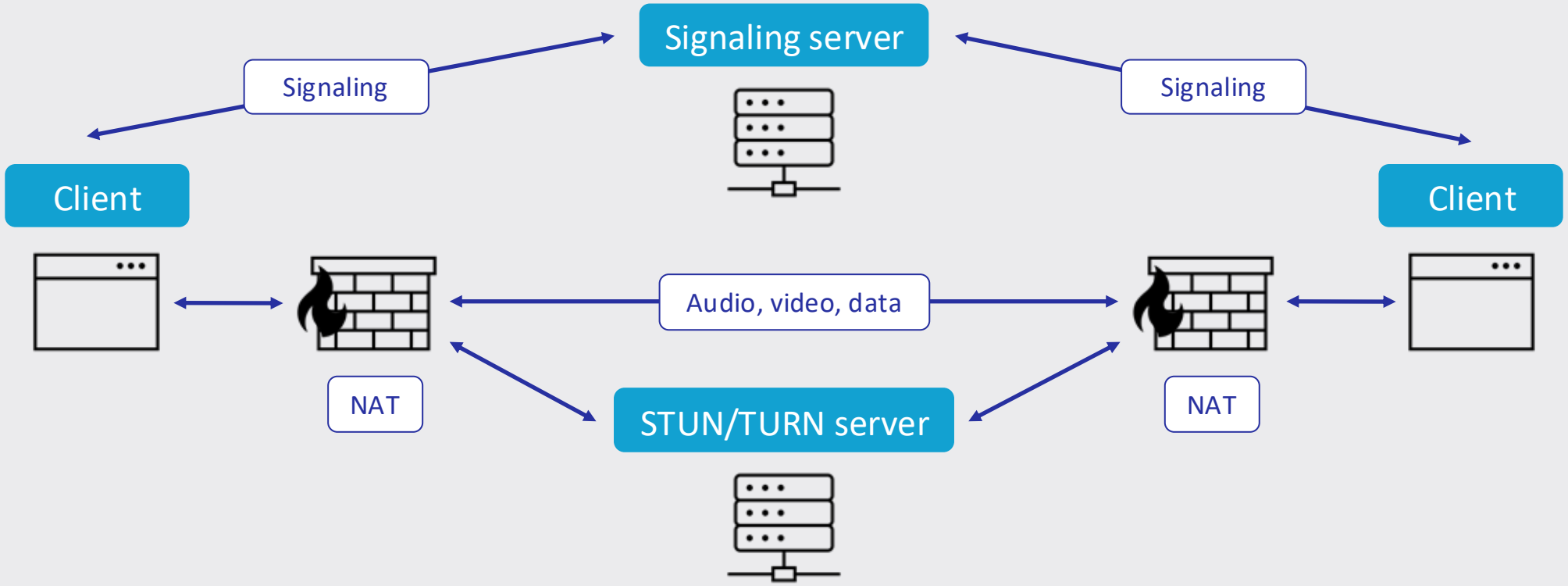
Extra features over plain WebSockets:

- Long polling fallback
  - In case WebSocket connection cannot be established
- Automatic reconnection
  - Heartbeat mechanism, which periodically checks the status of the connection
- Packet buffering
  - packets are automatically buffered when the client is disconnected, and will be sent upon reconnection
- Acknowledgements
  - Get a response if the event was successful
- Broadcasting
  - Send events to all connected clients or to a subset of clients
- Multiplexing
  - Namespaces allow you to split the logic of your application over a single shared connection
- WebTransport support
  - Able to use the newer protocol WebTransport

Demo

WebRTC

- Peer-to-peer communication protocol, mainly for audio/video/data
- Can bypass servers after initial signaling (though STUN/TURN may be used)
- Optimized for low-latency, real-time streaming



WebRTC

- True peer-to-peer, reduces server bandwidth usage
- Designed for real-time media (audio, video) and data
- Very low latency, ideal for conferencing and live interaction
- Supports NAT traversal with STUN/TURN servers

- Much more complex setup (signaling, ICE, TURN/STUN)
- Harder to scale for large groups (needs MCU/SFU servers)
- Peer-to-peer may fail in strict network/firewall environments
- Higher resource usage for multiple connections

Use cases: Video calls, P2P file sharing, multiplayer games

Demo

Very, very short recap 

- **(Long) Polling**: Simple, but inefficient
- **Server-Sent Events (SSE)**: Best for lightweight server → client updates
- **WebSockets**: Best for real-time two-way communication
- **WebRTC**: Best for peer-to-peer media/data streaming

Comparison Table

Feature	(Long) Polling	SSE	WebSockets	WebRTC
Direction	Client → Server	Server → Client	Bidirectional	Peer-to-Peer
Protocol	HTTP	HTTP	WS (TCP)	UDP (SRTP, DTLS)
Binary Support	No	No	Yes	Yes
Complexity	Low	Low	Medium	High
Latency	High	Very Low	Very Low	Depending on distance between clients
Use Cases	Legacy apps	Feeds/alerts	Chat/games/tools	Calls/files/games

WebTransport (new kid in town)

- Modern replacement for both WebSockets / WebRTC
- Built-in encryption
- Both TCP/UDP
- Faster / more efficient in every aspect compared to other real-time technologies

- Limited browser / server support
- More complex to setup
- HTTP3 is required

Demo

APEX Message Service

AMS makes WebSockets easy!

- Add real-time updates to your application
- Quickly start sending and receiving messages
- Sample application with lots of different use cases
- Fully integrated in APEX:
  - Dynamic Action and Process plug-in
  - PL/SQL API

More info: https://www.united-codes.com/products/apexmessageservice/

Thank you!

vito.vanhecke@united-codes.com

kevin.thyssen@united-codes.com