Pushing

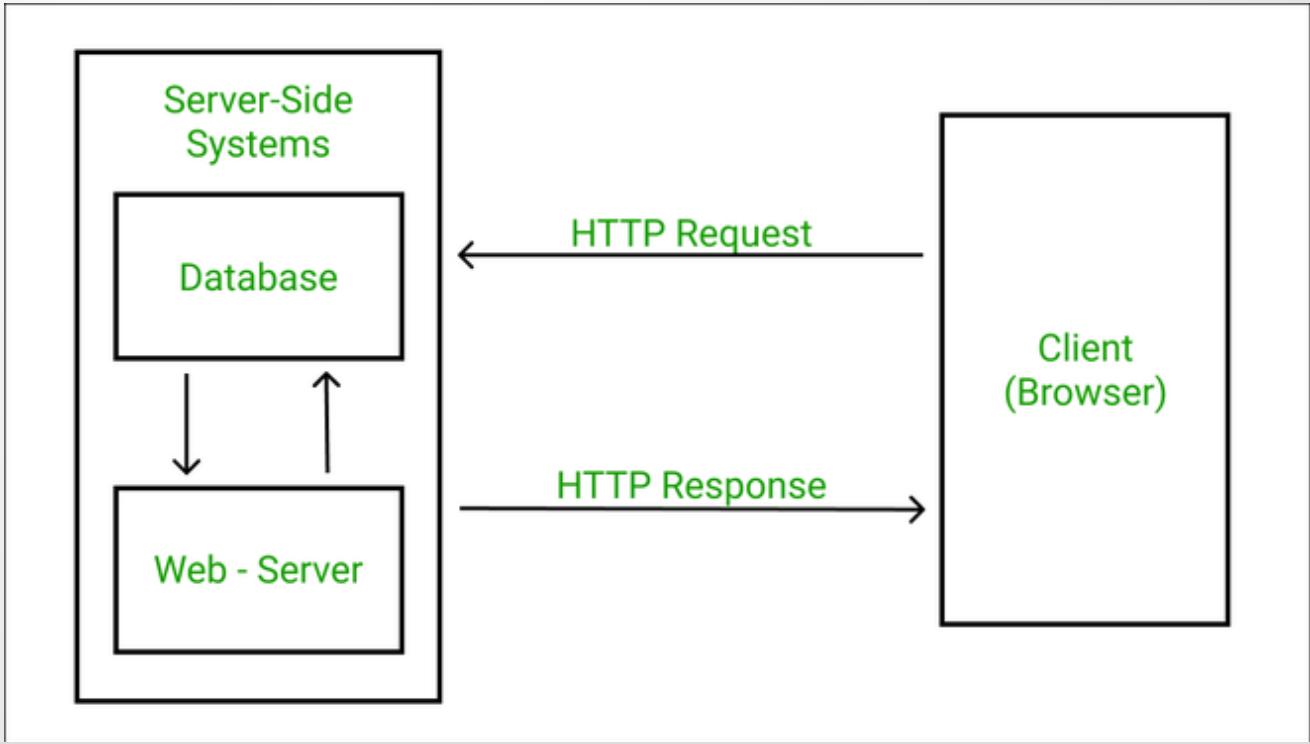
Webservers

Request-Response

Client-Server Communication

Client-Server

- Client initiiert immer die Kommunikation mit einem Request.
- Der Server kann keine Kommunikation initiieren



Alternatives

- Polling
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- Websockets
- Streaming
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- gRPC
- Web-Hooks
- REST-Hooks
- Libraries using mixed approaches

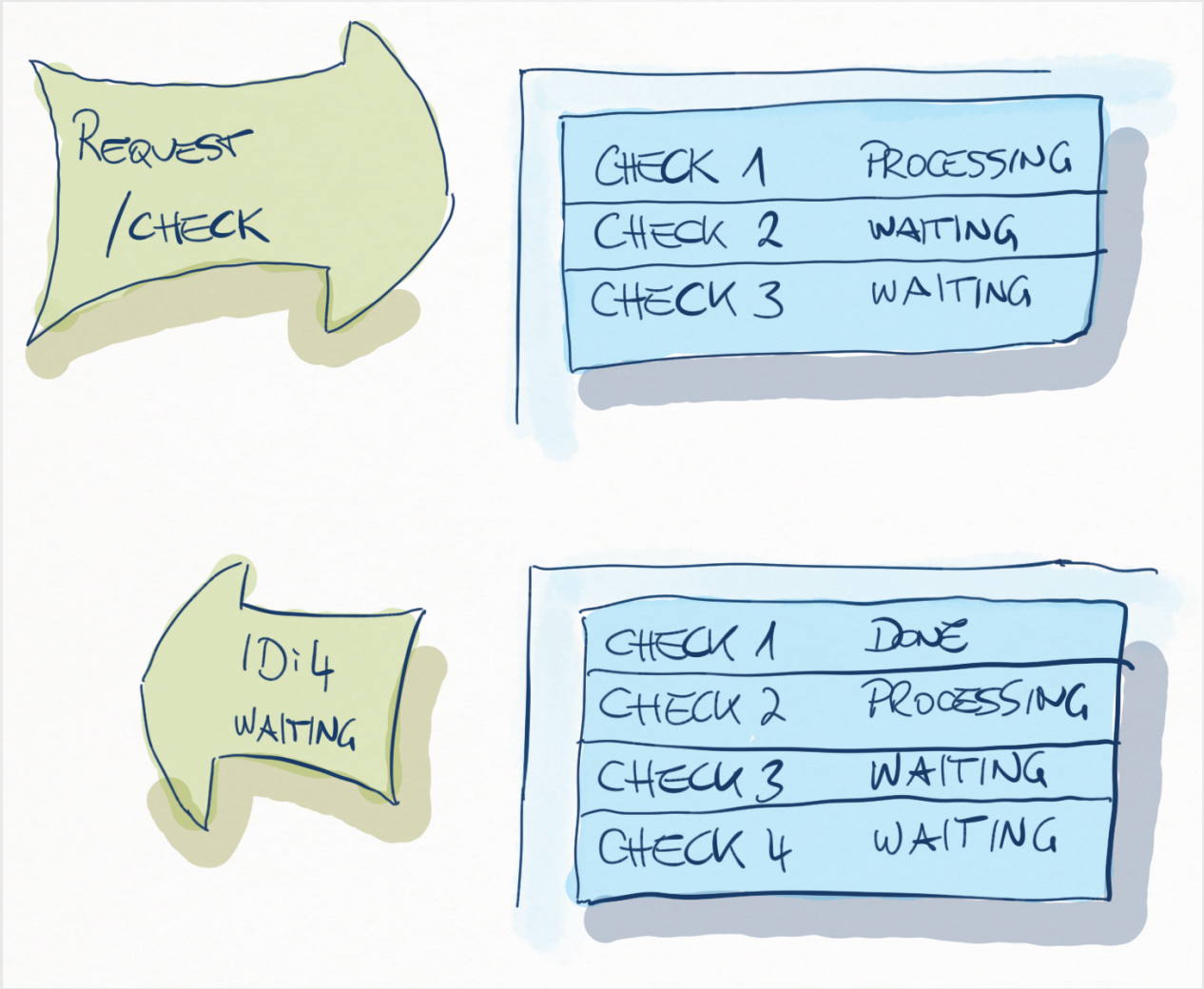
Setup

For all those approaches we have a resource, say a check, residing on a server and a client requesting that resource.

The problem now is, that, when generating this resource by creating a check-resource (telling the server to check something), the returned object may not be the final return-object.

It may contain a state set to 'processing' or something similar, telling the client that he has to re-request that resource until its state is 'done'.

Setup


REQUEST
/CHECK
CHECK 1
PROCESSING
CHECK 2
WAITING
CHECK 3
WAITING
ID:4
WAITING
CHECK 1
DONE
CHECK 2
PROCESSING
CHECK 3
WAITING
CHECK 4
WAITING

Polling

- **Polling**
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- Websockets
- Streaming
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- gRPC
- Web-Hooks
- REST-Hooks
- Libraries using mixed approaches

Polling

When doing polling we just check our check-resource periodically and work with the returned object.

If it still contains the state 'processing' we'd simply start another request later on until we get our return-object with the desired state.

Polling

Upsides

- No additional infrastructure needed.

Downsides

- Raises the number of concurrent requests your server has to handle drastically.
- Your answer is 'time_to_wait + polling_interval' seconds away, in the worst case.
- It's up to the client to decide how often to poll. If the developers chose this number too low they'd kill your server.

Long Polling

- Polling
  XmlHttpRequest (XHR), JsonP
- **Long Polling**
  XmlHttpRequest (XHR), JsonP
- Websockets
- Streaming
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- gRPC
- Web-Hooks
- REST-Hooks
- Libraries using mixed approaches

Long Polling

This method is like polling, but, as the server, you'd actually not return the object until you're sure its state is 'done'.

So when the client asks for the resource the first time, you wait for the resource to become available and turn 'done' before responding to the request.

When the connection (HTTP) times out, the client opens another one.

Long Polling

Upsides

- No additional infrastructure needed.
- Disconnects are handled gracefully automatically.
- Your answer arrives instantly after becoming available.

Long Polling

Downsides

- Keeps a connection open for as long as the resource takes to become available. So you may be running out of threads on Apache or affiliated servers.
- On the server the thread is connected to the connection.
  The thread you're working with when generating the resource is occupied. That means no real asynchronicity because killing your thread means killing the request.

Websockets

- Polling
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- **Websockets**
- Streaming
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- gRPC
- Web-Hooks
- REST-Hooks
- Libraries using mixed approaches

Web Sockets

WebSocket is a communications protocol for a persistent, bi-directional, full duplex TCP connection from a user’s web browser to a server.

A WebSocket connection is initiated by sending a WebSocket handshake request from a browser’s HTTP connection to a server to upgrade the connection.

After that, it's not an HTTP connection any longer, but a binary bi-directional connection as long as no peer closes it.

Web Sockets

Upsides

- The client is immediately informed about the resource becoming available, or even delivers it directly.
- Bi-directional streaming works.

Downsides

- Additional infrastructure.
- You'd have to write a language the main application-server and the websockets server may talk into each other.

Web Sockets

The application server would have to tell the websockets server to open a socket to the address X with the purpose of Y and wait for further notification. Then the application server would trigger that notification, possibly even containing the return object, when it's done with its asynchronous task. The websockets server could close the socket now.

Most webservers can do that now on their own. That doesn't mean though that they are the most performant or resource-efficient at that.

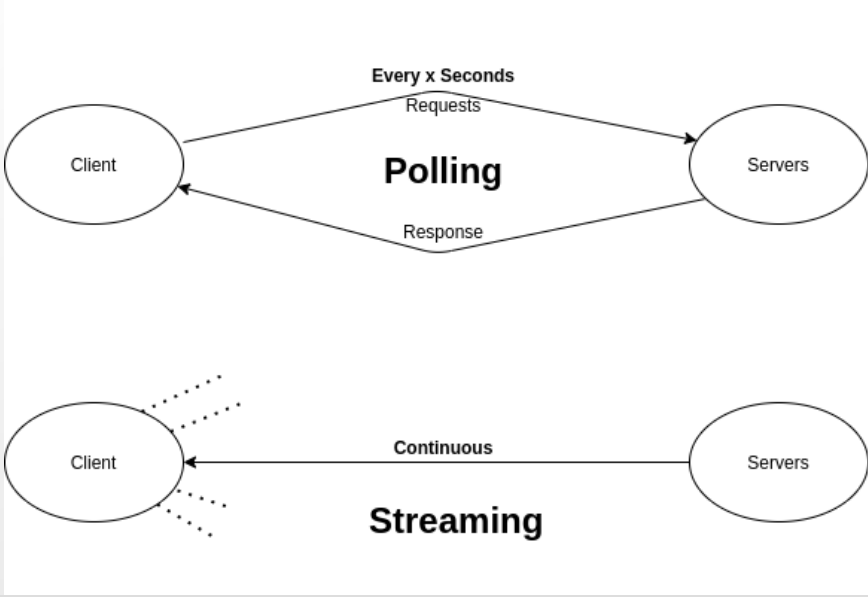
Streaming

- Polling
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- Websockets
- **Streaming**
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- gRPC
- Web-Hooks
- REST-Hooks
- Libraries using mixed approaches

Streaming

Is done using sockets.

Sockets are kept open until a peer closes it.



Streaming

Upsides

- Instant update.
- No request-storm.

Streaming

Upsides

- Uses XHR streaming to transport messages over HTTP.
- Event-based; no polling required to intercept messages.

Streaming

Downsides

- Can be more difficult to set up.
- Every user requires a connection.
- More resource-usage on both sides.
- Connections must be kept open at all times.
- Can be a bit hard to coordinate with the request/response based code.
- When client cannot keep up, the data builds up in the buffer of the connection (server piles up messages in the open connection)

SSE

- Polling
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- Websockets
- Streaming
  XmlHttpRequest (XHR), JsonP
- **Server Sent Events** (SSE)
- gRPC
- Web-Hooks
- REST-Hooks
- Libraries using mixed approaches

SSE

Server-Sent Events (SSE) are based on Server-Sent DOM Events.

Browsers can subscribe to a stream of events generated by a server using the EventSource interface, receiving updates whenever a new event occurs.

Uses XHR Streaming

SSE

Downsides

- Limited concurrent connections (like in streaming-approach). Limit is 6 by browser.
- Data format limitations. (UTF-8 only, no binary)
- Mono-Directional. Designed to contact the client. Client cannot send data.

gRPC

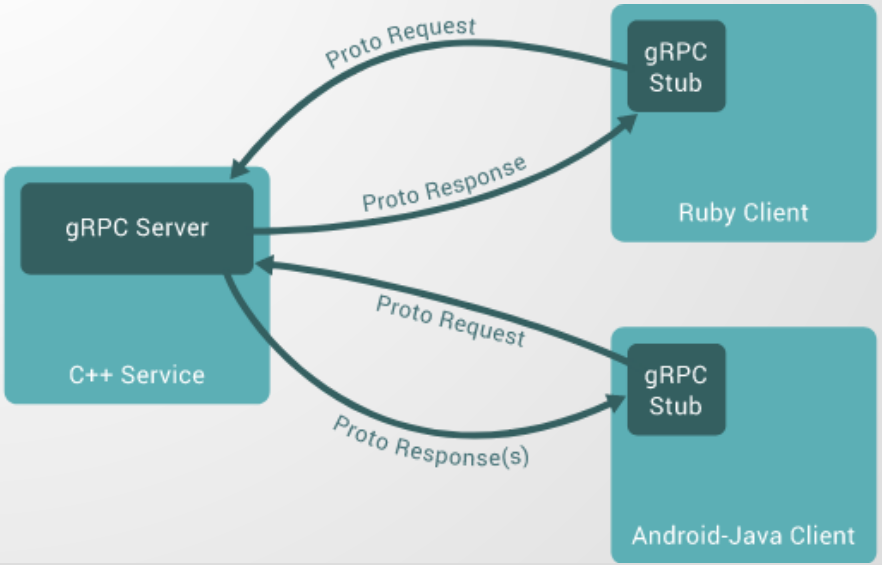
- Polling
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- Websockets
- Streaming
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- **gRPC**
- Web-Hooks
- REST-Hooks
- Libraries using mixed approaches

gRPC

A high performance, open-source universal RPC framework developed by Google.

Used by Google to connect their microservices.

Can be used on the last mile (backend to client) as well.



gRPC

Upsides

- Uses Protocol Buffers for a small message-footprint.
- Stubs are available in many different languages.
- Works across platforms.
- Simple service-definition.
- Bi-directional streaming available.

gRPC

Downsides

- Uses a separate language for service definitions.
- Primarily designed for service-to-service communication, not for the last mile.
- Heavily relies on code generation.
- Just unary and server-side streaming for JavaScript clients at the moment.

Web-Hooks

- Polling
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- Websockets
- Streaming
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- gRPC
- **Web-Hooks**
- REST-Hooks
- Libraries using mixed approaches

Web-Hooks

Classic, or user-managed, webhooks get closer to the ideal event notification system. The server POSTs a payload to some user-defined URL.

There is almost zero overhead on the client side.

They just handle the POST requests like normal.

Web-Hooks

Upsides

- No additional infrastructure needed.

Downsides

- Only for server - server communication (what's the URL a client, like a browser, would enter into the subscription?).
- User has to provide a callback-URL.
- Your system depends on the speed of a foreign service while waiting for the response of the foreign server.
- Many dead callback URLs may seriously compromise the performance of your system when doing many callbacks at once.

REST-Hooks

- Polling
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- Websockets
- Streaming
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- gRPC
- Web-Hooks
- **REST-Hooks**
- Libraries using mixed approaches

REST-Hooks

Essentially this solution is webhooks with the addition that the subscriptions are exposed using a RESTful API.

So this mechanism may be used by application developers in order to subscribe or unsubscribe dynamically.

Method	Route	About
GET	/api/v1/subscription/	list subscriptions
POST	/api/v1/subscription/	create a subscription
GET	/api/v1/subscription/:id/	get a subscription
PUT	/api/v1/subscription/:id/	update a subscription
DELETE	/api/v1/subscription/:id/	delete a subscription

REST-Hooks

Upsides

- No additional infrastructure is needed.
- Fully REST-compliant solution.
- Intuitively comprehendible by any REST developer.

REST-Hooks

Downsides

- Only for server - server communication (what's the URL a client, like a browser, would enter into the subscription?).
- Your system depends on the speed of a foreign service while waiting for the response of the foreign server.
- Many dead callback URLs may seriously compromise the performance of your system when doing many callbacks at once.

Mixed

- Polling
  XmlHttpRequest (XHR), JsonP
- Long Polling
  XmlHttpRequest (XHR), JsonP
- Websockets
- Streaming
  XmlHttpRequest (XHR), JsonP
- Server Sent Events (SSE)
- gRPC
- Web-Hooks
- REST-Hooks
- **Libraries using mixed approaches**

Example: Pusher

Implements Fallbacks

- Web Socket
- Flash Socket
- XHR
- SSE
- Iframes (infinitely long page in an IFrame)
- JasonP

How it connects

- Test unsafe methods
- Develop a strategy in which order to test them and which in parallel - one after another is too slow
- Memorize them in browsers local storage

Problems

Problems

There are surprisingly many things that can go wrong when trying to establish a real-time connection:

- a browser might not support native WebSockets,
- Flash might not be installed or can just be blocked,
- firewalls can block specific ports (e.g. 843 for Flash),

Problems

- some proxies might not support WebSocket protocol,
- other proxies simply don’t like persistent connections and terminate them.

All these problems manifest themselves in different ways – some are easy to resolve, some just take a significant amount of time to get noticed.

References

- https://medium.com/@reemshakes/http-request-response-flow-for-dummies-a0f52af83af3
- https://blog.pusher.com/how-we-built-pusher20-part-1/
- https://ably.com/blog/websockets-vs-sse
- http://stackoverflow.com/questions/36086837/long-polling-in-a-spring-rest-application
- http://resthooks.org/docs/alternatives/
- http://resthooks.org/
- https://github.com/gimite/web-socket-js
- https://pusher.com
- https://bloggeek.me/websockets-http2/
- https://webrtchacks.com/an-intro-to-webrtcs-natfirewall-problem/
- https://github.com/realtime-framework/ChromePushNotifications

References

- https://www.ably.io
- https://www.leggetter.co.uk/real-time-web-technologies-guide/
- https://grpc.io/
- https://www.geeksforgeeks.org/polling-and-streaming-concept-scenarios/