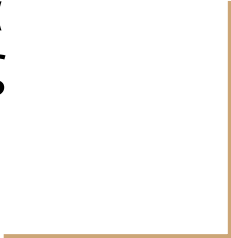
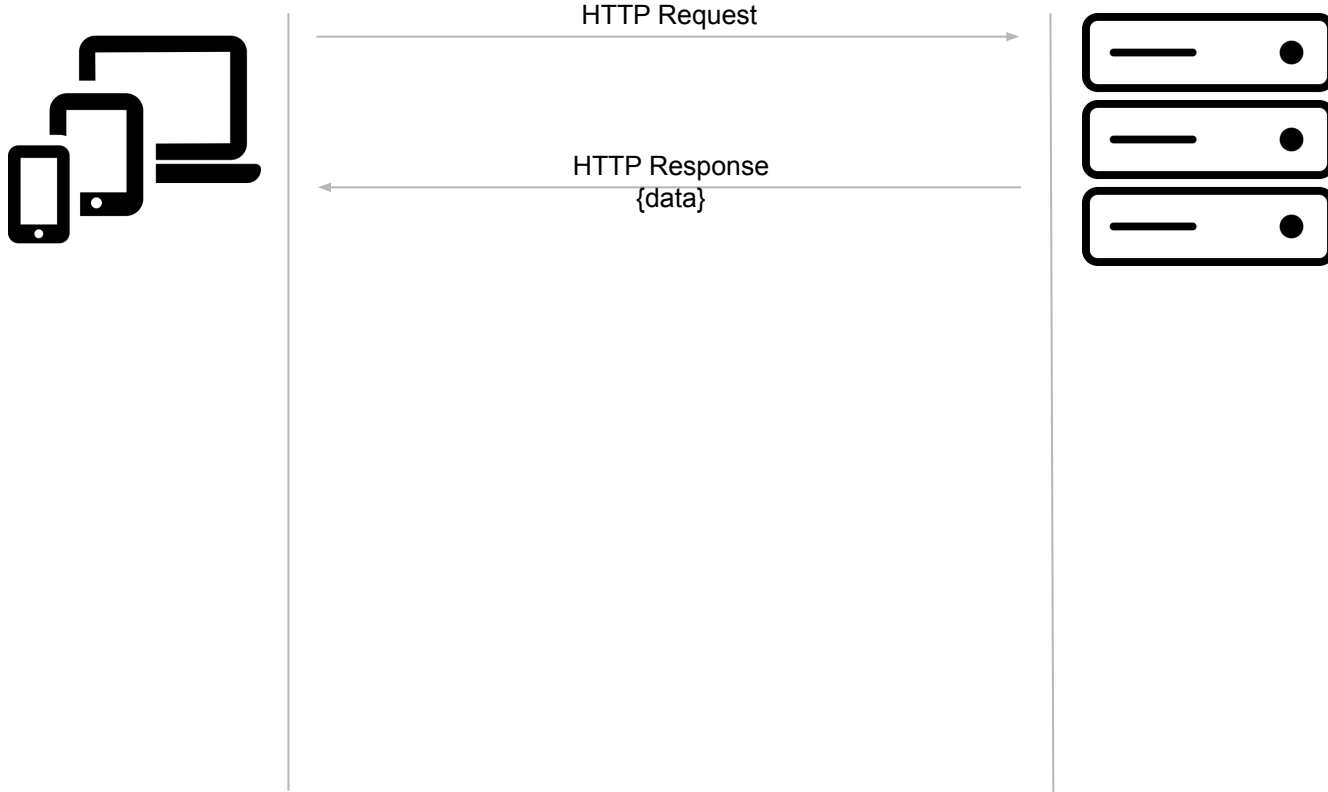


Τεχνολογίες Εφαρμογών Ιστού

Πρωτόκολλα
επικοινωνίας



Standard HTTP request-response cycle



Standard HTTP

- HTTP request-response cycle
 - Client sends request to server and server responds
 - Data is **pulled** from the server
- What happens when the server needs to initiate the conversation (**push** data)?
 - Chat
 - Games
 - Feeds
 - ...

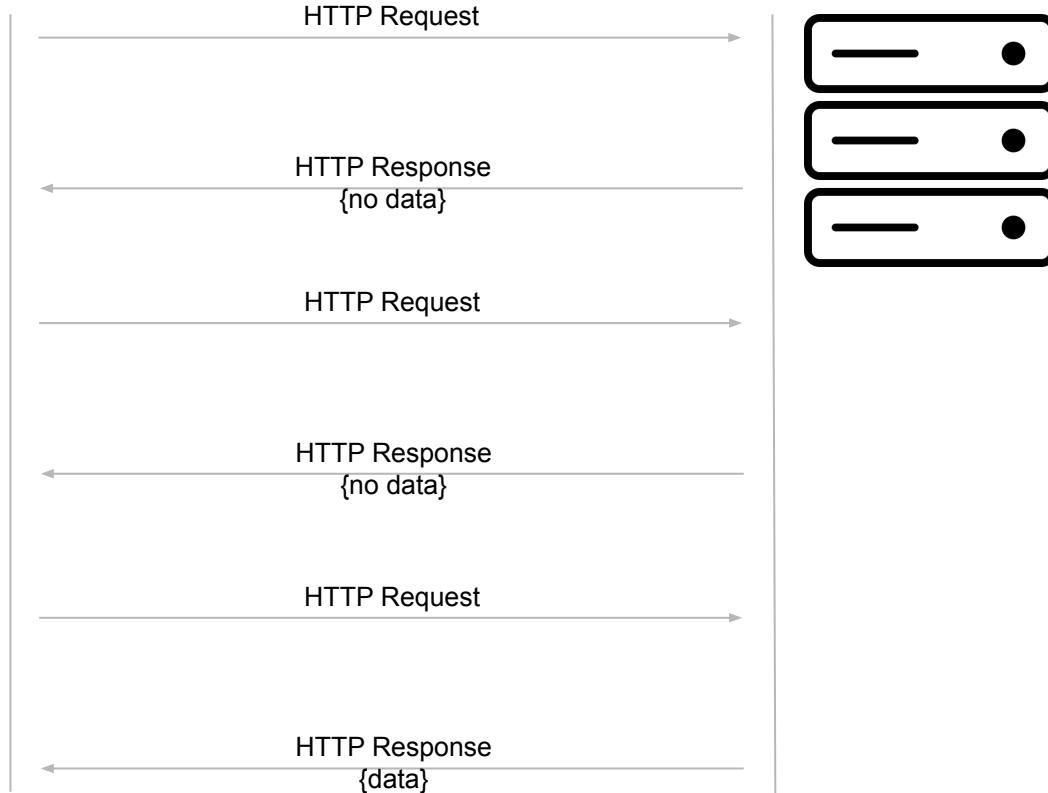
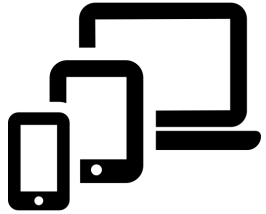
Full duplex

- Full-duplex or double-duplex system, allows communication in both directions simultaneously
- A half-duplex system provides communication in both directions, but only one direction at a time (not simultaneously)

Communication options

- Repeated requests (polling)
 - Client periodically asks server if there's anything it would like to say
- Dual roles (P2P)
 - Physical server also implements a client and the client implements a server too
- HTTP Long Polling
 - Clients sends "long" requests which remain open until the server responses
- Server-Sent Events (SSE)
 - Keeps the connection open using XHR streaming (over XMLHttpRequest)
- Message broker
 - Middle component facilitating message passing between app components
- Websockets
 - Thin layer over TCP for full-duplex communication
- ...

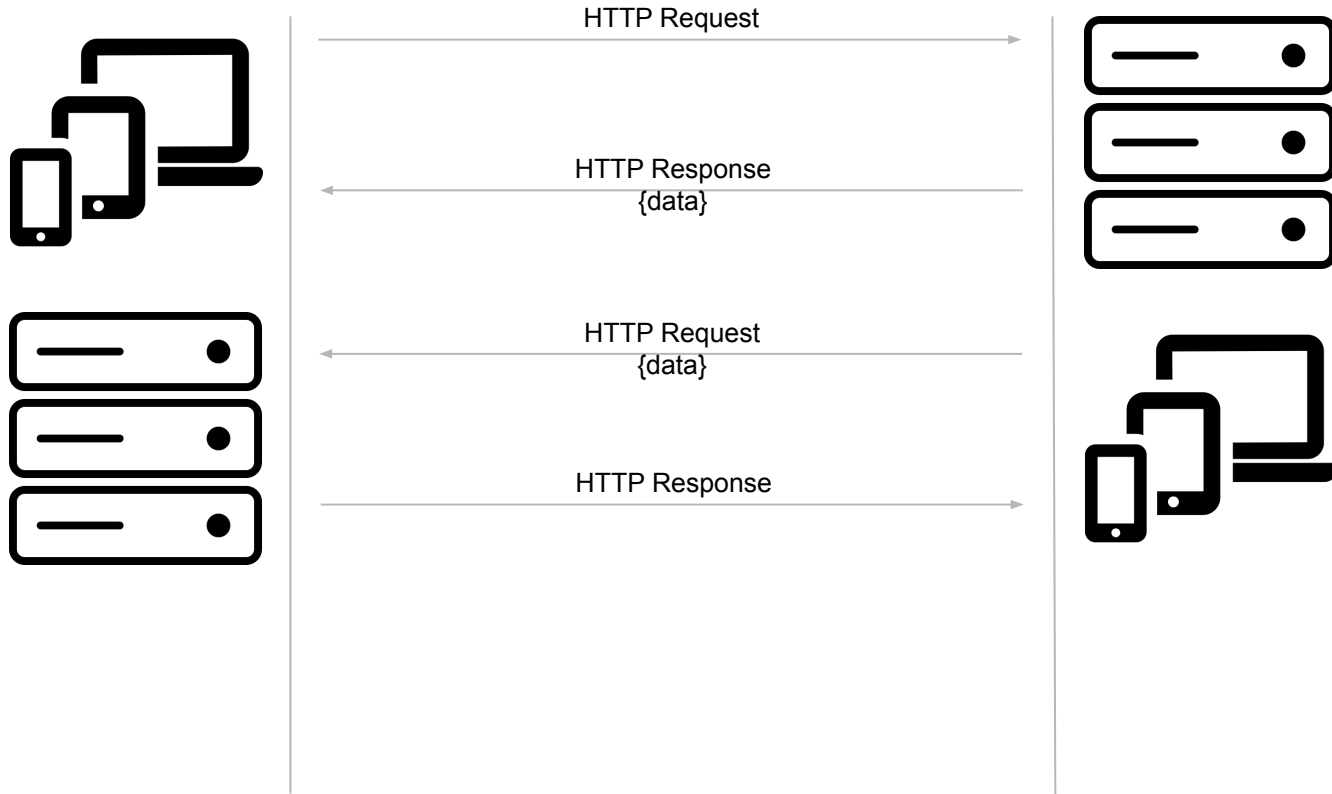
Repeated requests



Repeated Requests

- Pros
 - Few changes in the code, architecture, protocols
 - Still using HTTP
- Cons
 - Huge overheads
 - Each new incoming connection must be established
 - The HTTP headers must be parsed
 - A query for new data must be performed
 - A response (usually with no new data to offer) must be generated and delivered
 - The connection must then be closed and any resources cleaned up

Dual roles (P2P)



Dual roles (P2P)

- Pros
 - Still using HTTP
- Cons
 - Demanding on the client side
 - needs resources
 - Permissions
 - special client-side software
 - not supported by browsers
 - Complex synchronization problems

Hypermedia as the Engine of Application State (HATEOAS)

- HATEOAS is a constraint of the REST application architecture
- A user-agent makes a HTTP request of a REST API through a simple URL.
- All subsequent requests the user-agent may make are discovered inside the responses to each request.
- The media types used for these representations, and the link relations they may contain, are standardized.
- In this way, RESTful interaction is driven by hypermedia, rather than out-of-band information.

HATEOAS example

- Request

GET /accounts/12345 HTTP/1.1

- Response

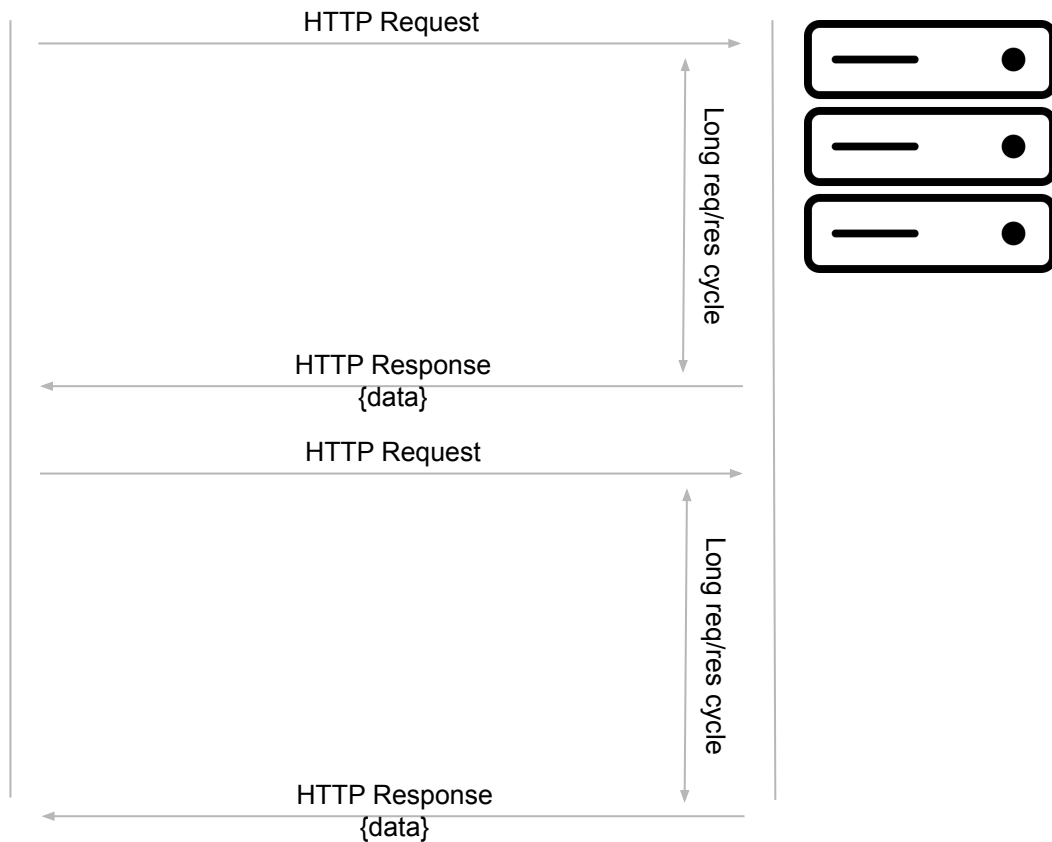
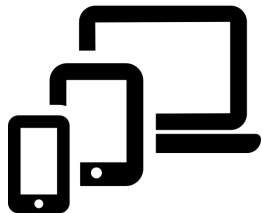
HTTP/1.1 200 OK

Content-Type: application/vnd.acme.account+json

Content-Length: ...

```
{
  "account": {
    "account_number": 12345,
    "balance": {
      "currency": "usd",
      "value": 100.00
    },
    "links": {
      "deposit": "/accounts/12345/deposit",
      "withdraw": "/accounts/12345/withdraw",
      "transfer": "/accounts/12345/transfer",
      "close": "/accounts/12345/close"
    }
  }
}
```

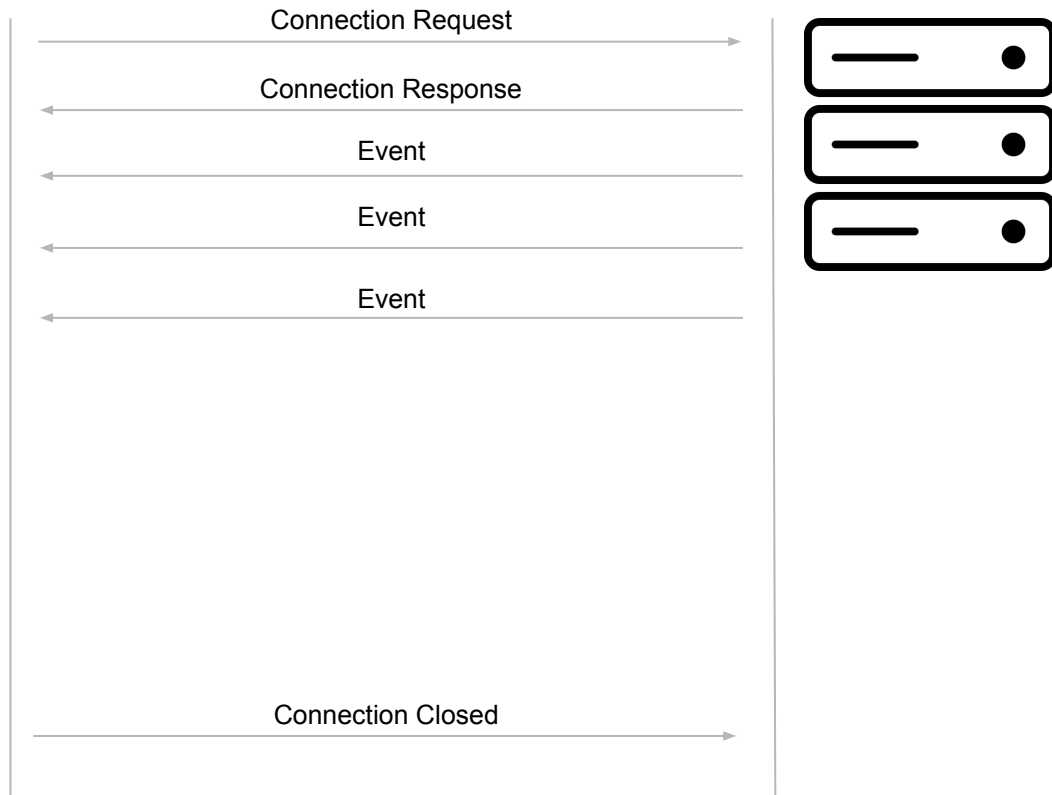
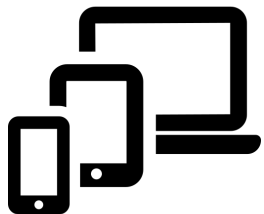
Long Polling



Long Polling

- Pros
 - Ease of use: Implemented on the back of XMLHttpRequest
 - Little changes in the architecture and code
- Cons
 - Demanding on the server side (Server keeps the connection open)
 - Potential synchronization issues (imagine two requests remaining open at the same)

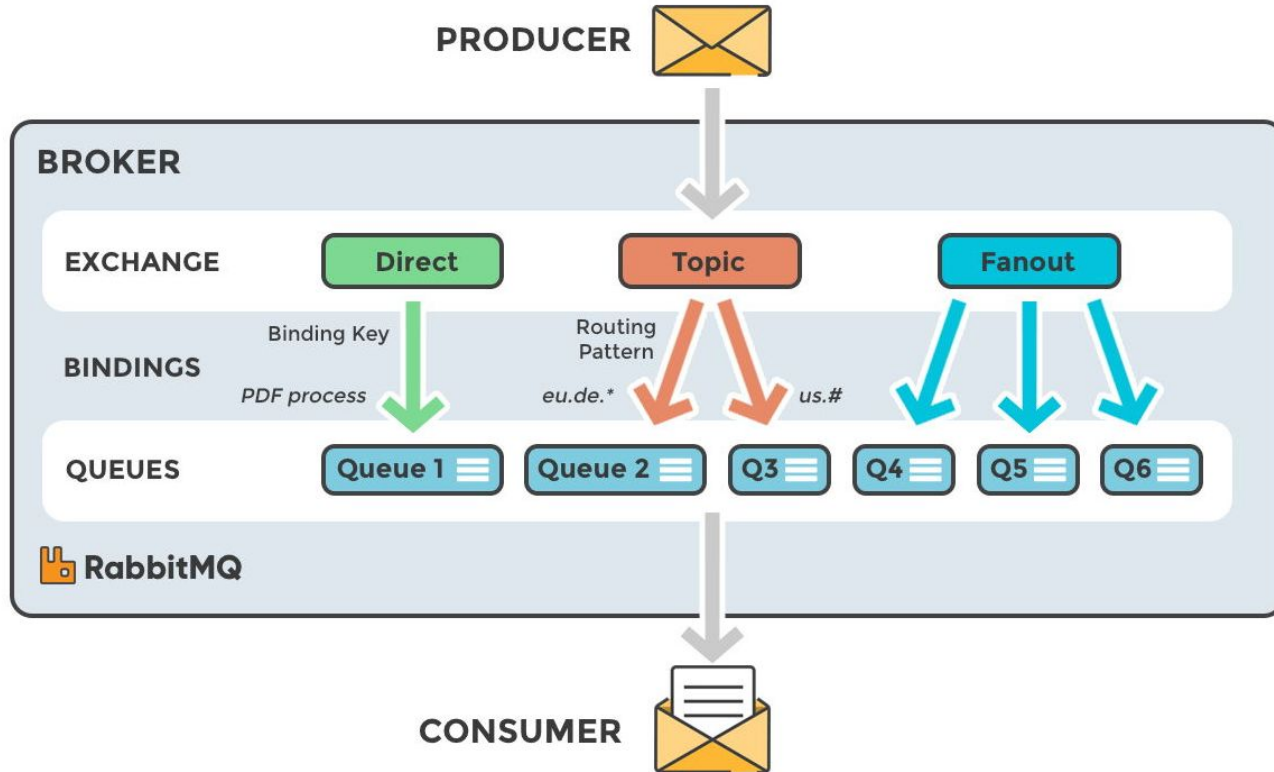
Server-Sent Events (SSE)



Server-Sent Events (SSE)

- Pros
 - Minor changes and overheads
- Cons
 - Doesn't facilitate load balancing and scaling
 - Can't resume from broken communication links
 - Essentially it is mono-directional !

Message Broker



Message Broker

- An intermediate component is receiving messages
- ...and sends them out to all components that need them
- Server and client roles are indifferent
- Important roles are:
 - Message producers
 - Message consumers
- Still need for potentially full duplex communication of components with broker.
 - MQTT is one option
- Interesting protocol (not communication):
 - Publish-Subscribe (pubsub)

PubSub

- The broker can maintain multiple message queues
- A producer can publish messages to a queue
- All interested consumers can subscribe to that queue
- ...and the broker is pushing the data to them

Message broker

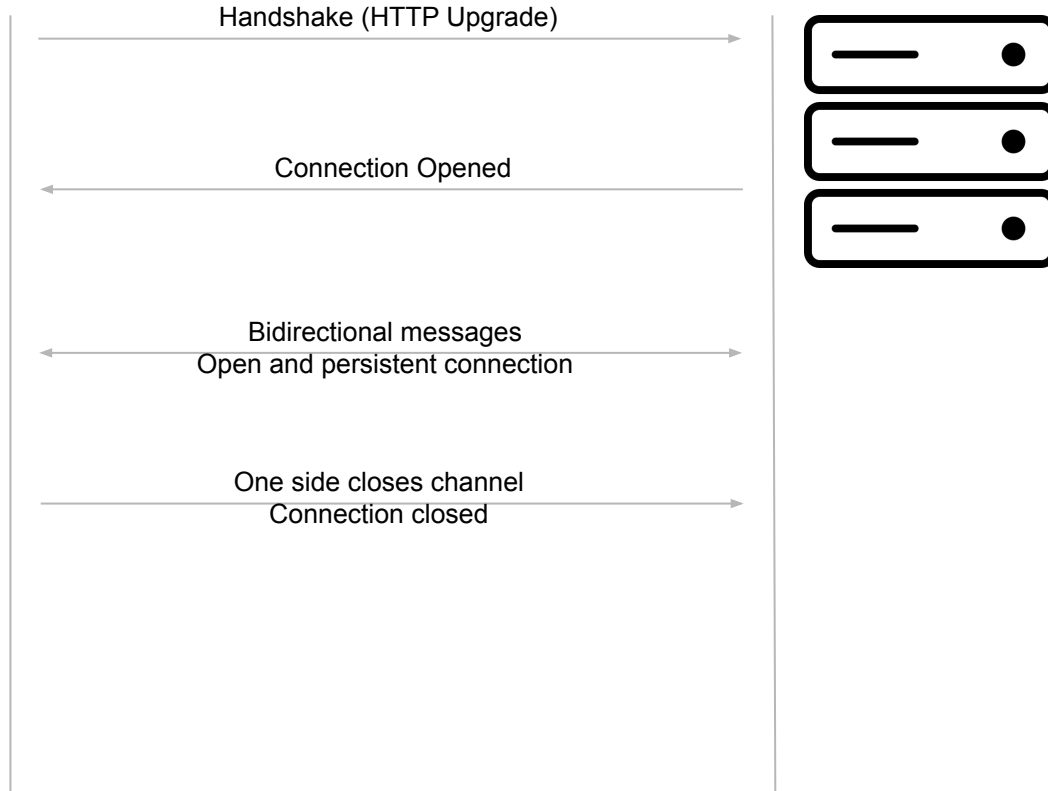
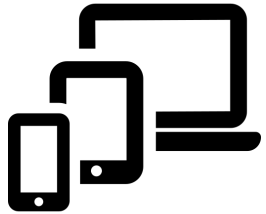
- Pros

- Low energy requirements for the individual components (good for IoT devices)
- Good for complex and dynamic workflows. You can wire components any way you like (good for microservices)
- Scalability: Any component that is spawned has to simply connect to the message broker and declare the queue it is interested to contribute or consume messages from (works well with autoscaling/load balancing services)
- Stable, open source tools to work with (e.g. RabbitMQ)

- Cons

- Requires a whole new architecture
- Overkill when there are a few components involved
- Still requires full duplex communication!

Websockets



Websockets

- WebSocket is a web technology providing full-duplex communications channels over a single TCP connection
- The WebSocket API is being standardized by the W3C, and the WebSocket protocol has been standardized by the IETF as RFC 6455
- Uses upgraded HTTP. Work with existing HTTP apps.

Websockets

- Pros
 - Fast
 - Low overhead
 - Event-based interfaces (<https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>)
- Cons
 - Fault tolerance (Doesn't recover connection in the case of errors)
 - Scaling is as difficult as in other solutions

Sources

- <https://en.wikipedia.org/wiki/WebSocket>
- <https://html.spec.whatwg.org/multipage/web-sockets.html>
- https://bohr.wlu.ca/hfan/cp476/16/notes/cp476_lecture18_websocket.pdf
- <https://www.ably.io/blog/websockets-vs-long-polling/>
- <https://www.ably.io/concepts/server-sent-events>
- <https://www.scaledrone.com/blog/websockets-vs-server-sent-events-sses/>
- <http://mqtt.org/>