31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Sie haben Post!

Mobiles polyglottes Messaging

Frank Pientka

MATERNA GmbH

MATERNA
Information & Communications

Sie haben Post

Mobiles polyglottes Messaging

Frank Pientka, Dortmund

Wer ist Frank Pientka?

Dipl.-Informatiker (TH Karlsruhe)

Principal Software Architect in Dortmund

iSAQB-Gründungsmitglied

heise.de/developer/Federlesen-Kolumne

Über 20 Jahre IT-Erfahrung
Veröffentlichungen und Vorträge

Frank Pientka
frank.pientka@materna.de
+49 (231) 5599 8854
+49 (1570) 1128854
www.materna.de

Agenda

Kommunikation verändert sich	Mobiles Messaging – Möglichkeiten & Heraus-forderungen	Messaging-Protokolle – MQTT & STOMP	Messaging-Produkte für MQTT & STOMP	Beispiel - JavaScript, Java, CLI	Fazit – wird möglich

Kommunikation verändert sich

Web 1.0: 1 Sender, viele Empfänger

Web 2.0: viele Sender, viele Empfänger

Mobiles Web, IoT, M2M:
immer & überall & jeder & jedes

Mobile ist die Zukunft (2014)

300 Mio

Verkaufte PCs

2,2 Mrd

Verkaufte Smartphones, Tablets

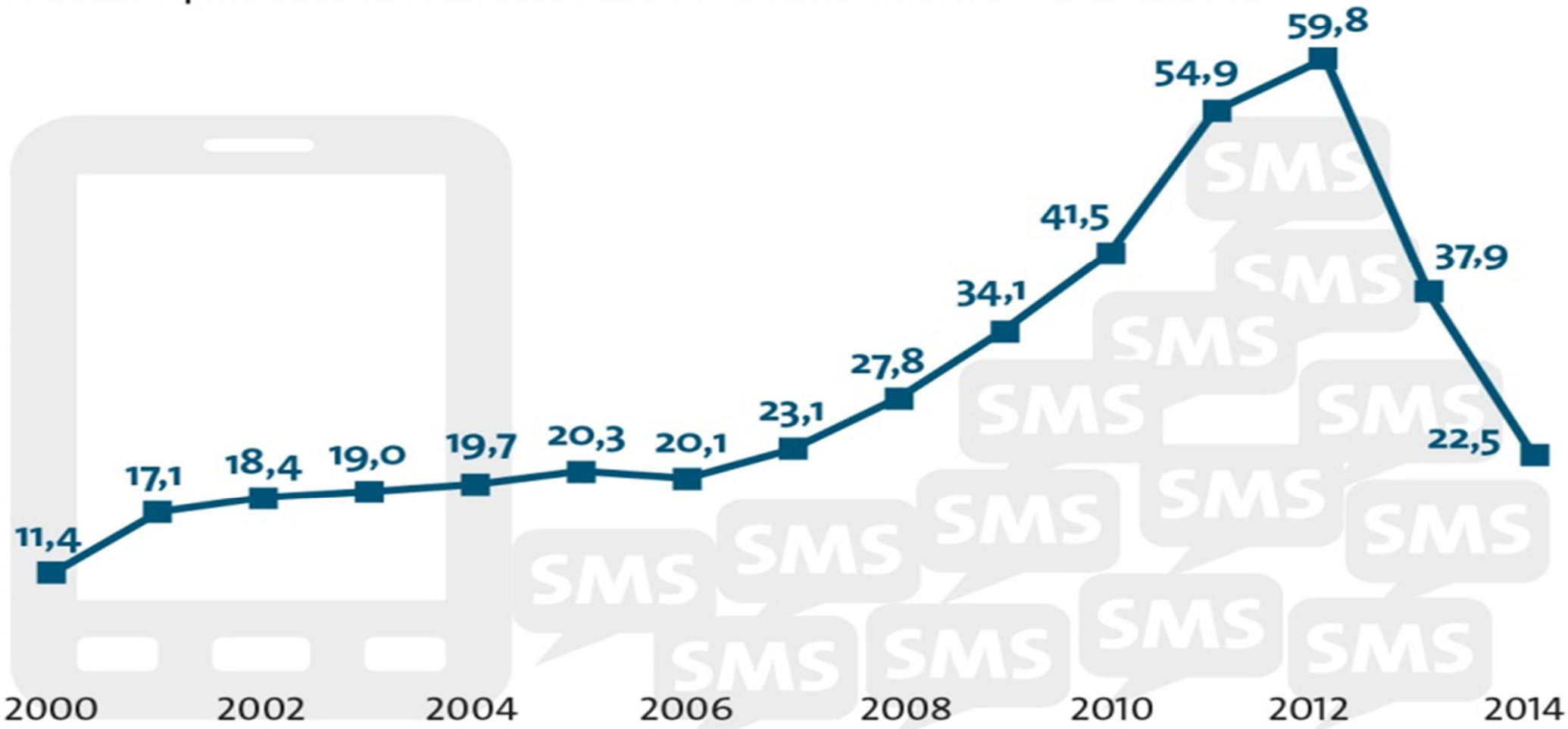
7,2 Mrd

Menschen

Quelle: Gartner

Die Talfahrt geht weiter

Zahl der pro Jahr versandten SMS in Deutschland in Milliarden



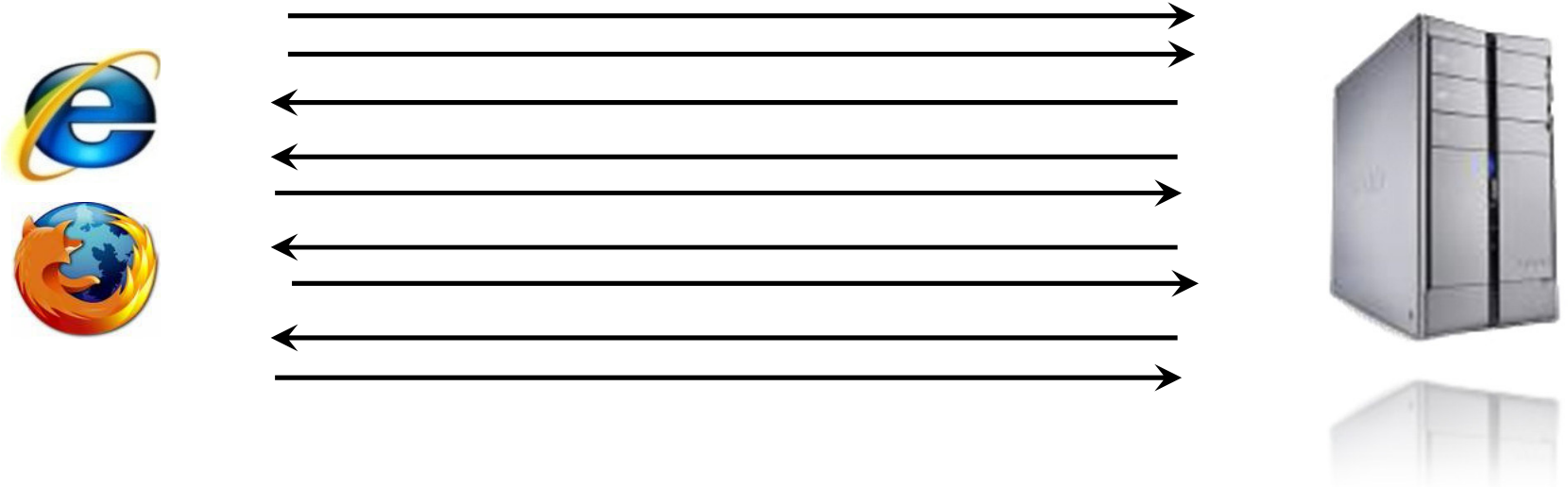
BITKOM

Quelle: Bundesnetzagentur

Das synchrone Web ..

…war gestern

Heute: Ereignisorientierte Verarbeitung



Mobiles Messaging
Quo vadis?

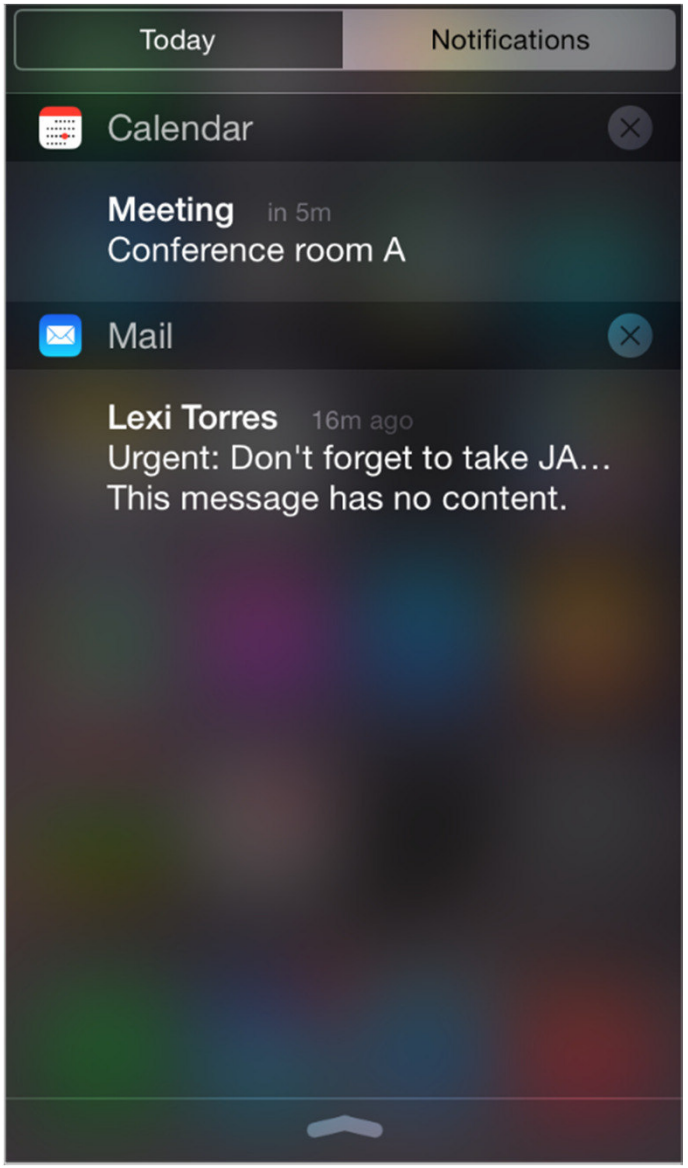
Agenda

Kommunikation verändert sich	Mobiles Messaging – Möglichkeiten & Heraus-forderungen	Messaging-Protokolle – MQTT & STOMP	Messaging-Produkte für MQTT & STOMP	Beispiel JavaScript, Java, CLI	Fazit – wird möglich

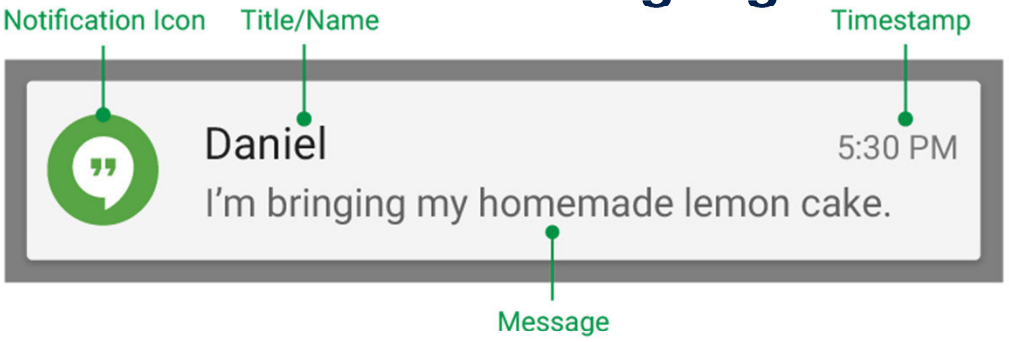
Mobile App-Arten und Technologien

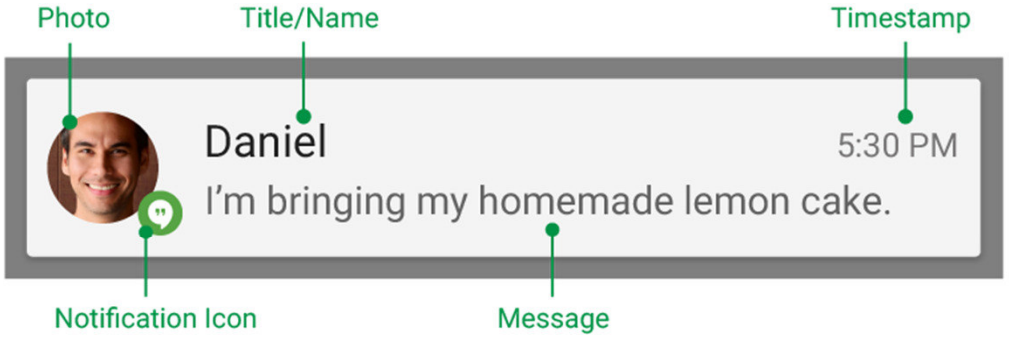
Web	Hybride	Native
STOMP MQTT	XMPP STOMP MQTT	XMPP STOMP MQTT
CSS3 JS HTML5	Xamarin Cordova	Objective-C Java

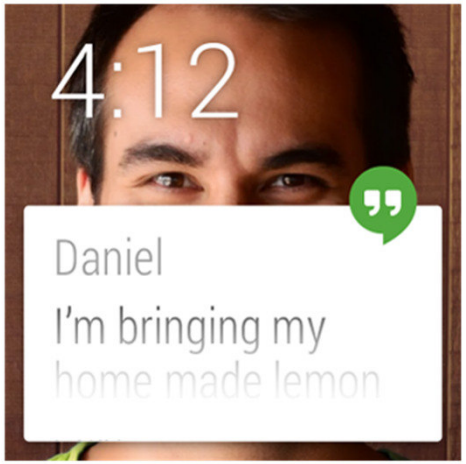
Benutzer Benachrichtigung - iPhone

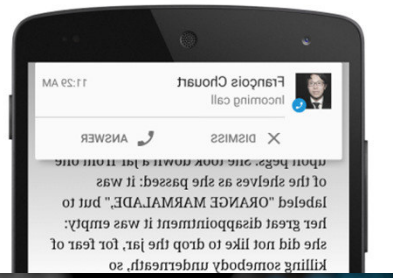


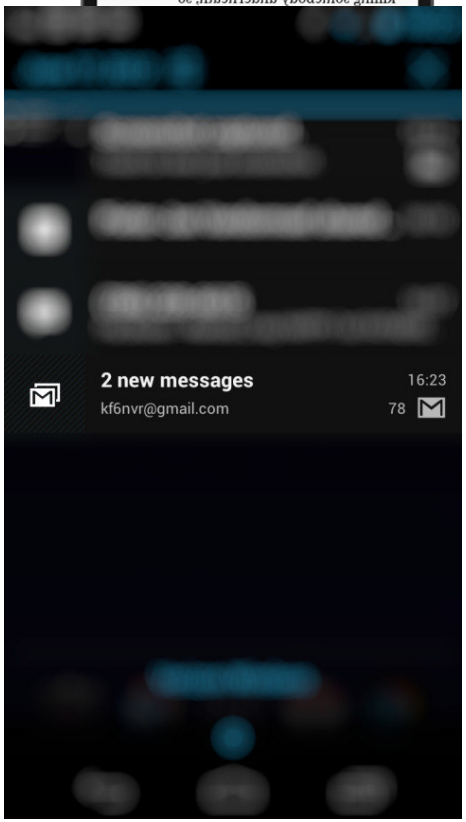
Benutzer Benachrichtigung - Android









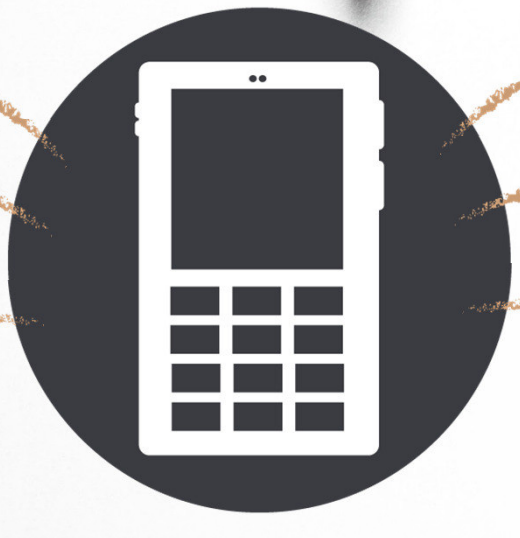


Herausforderungen mobile Kommunikation

Geringe Leitungskapazitäten

Innovation

Kurze Akkuzeiten



Instabile Verbindungen

Heterogenität

Offline-Fähigkeit

Herausforderungen

Heterogenität

Registrierung

QoS

Akku

Latenz

Innovation

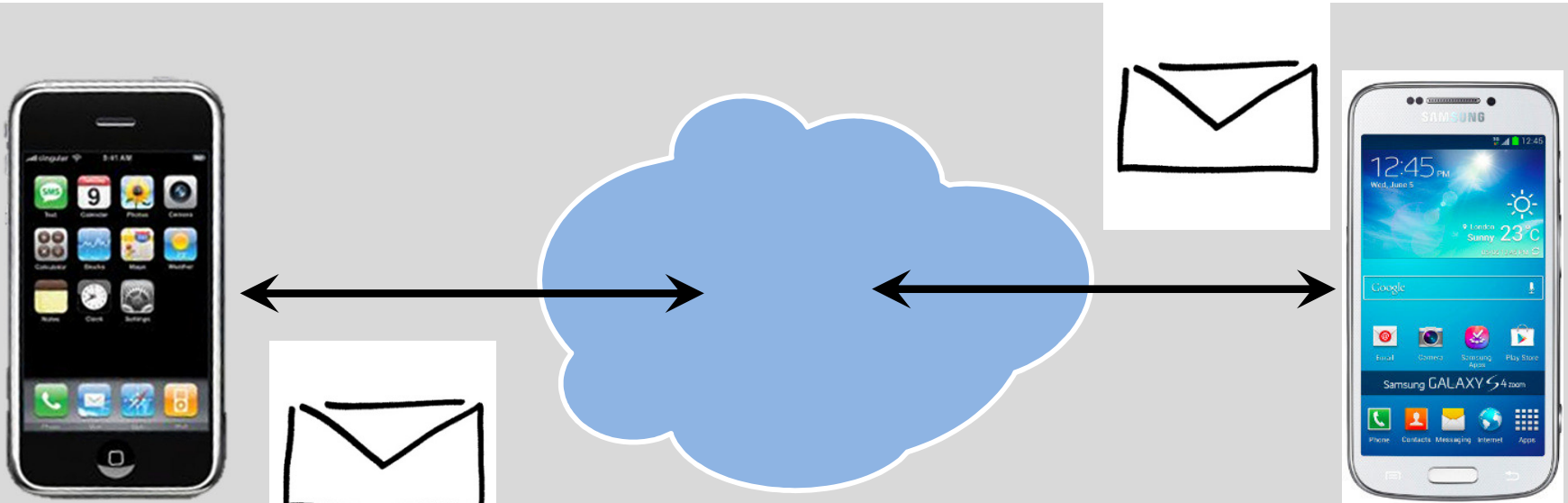
Notification

Robustheit

Sicherheit

Mobiles Messaging

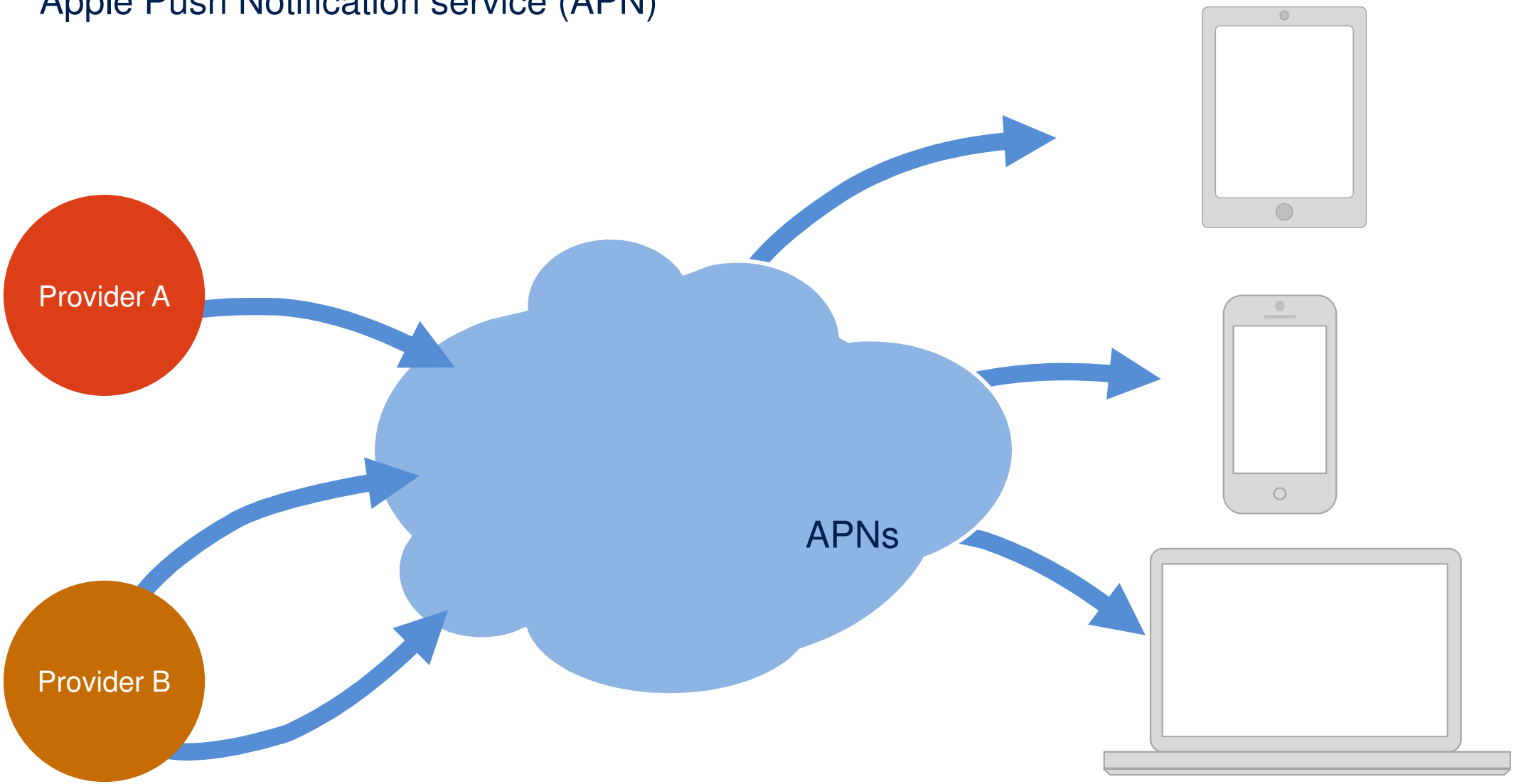
Mobile Nachrichten Überblick



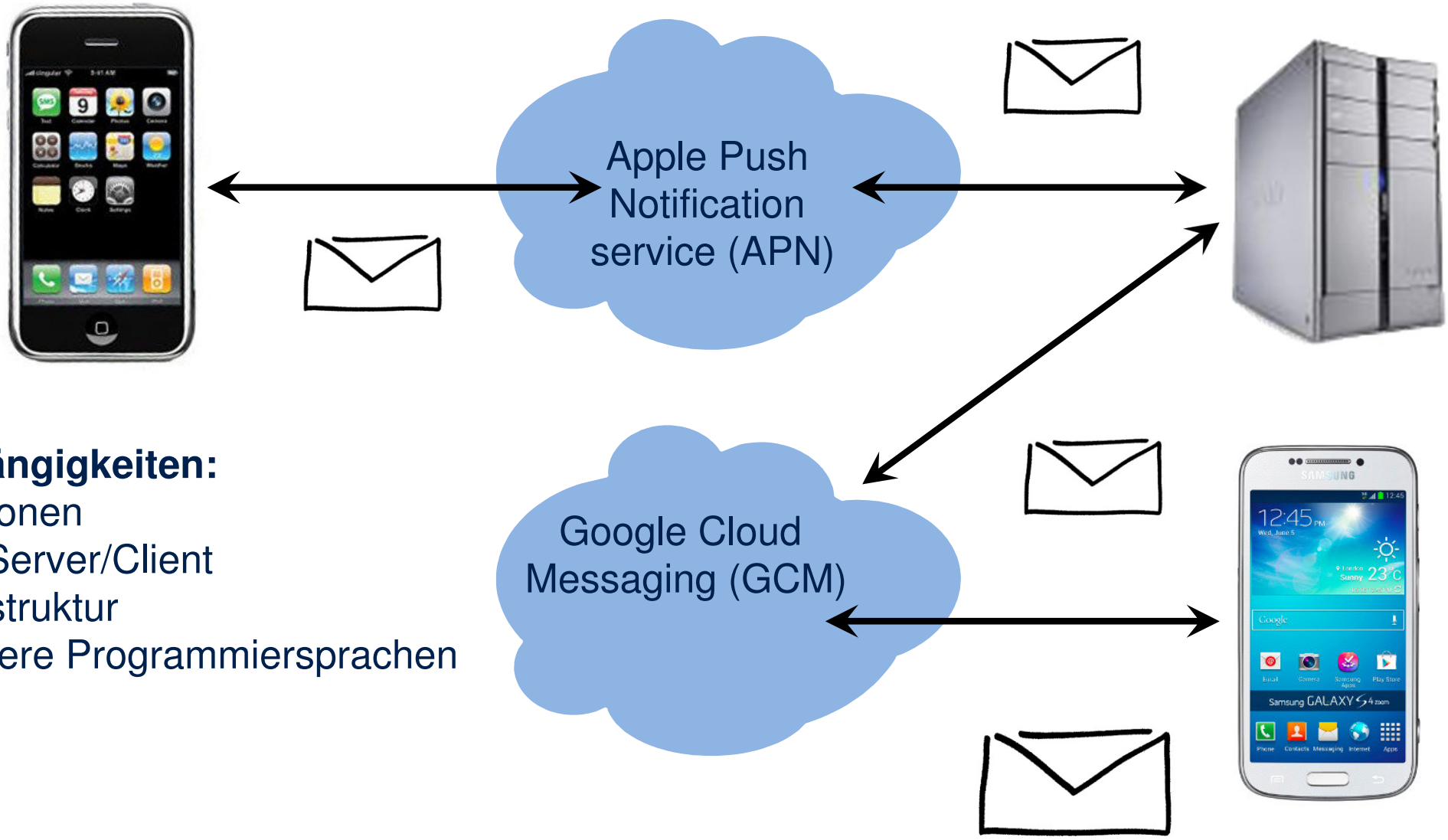
Email
SMS/MMS
Push-Nachrichten
....

Mehrere Anbieter, mehrere Nutzer

Apple Push Notification service (APN)

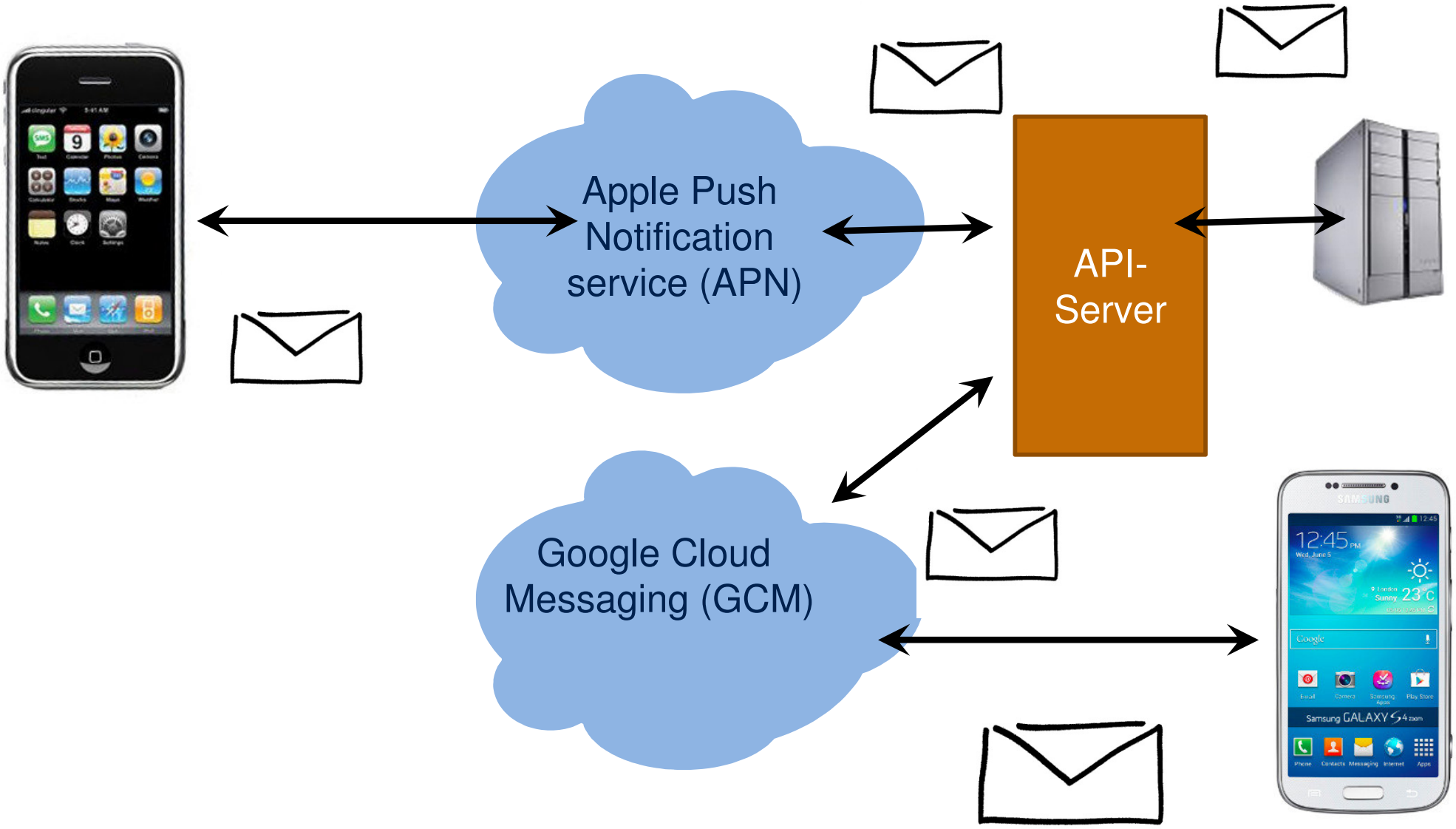


Mobile Nachrichten mit iOS und Android

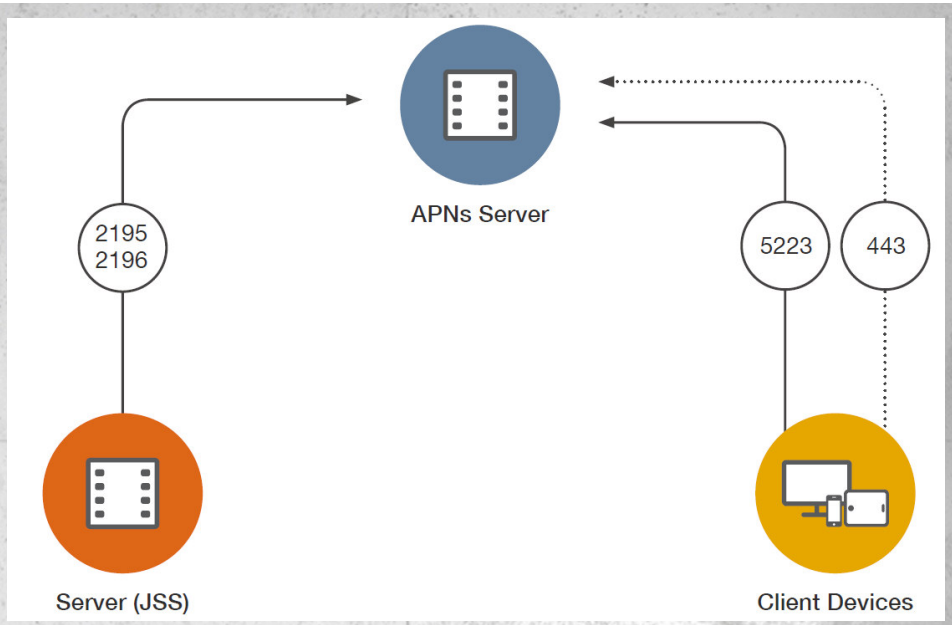


Abhängigkeiten:
Versionen
API-Server/Client
Infrastruktur
Mehrere Programmiersprachen

Mobile Nachrichten – vereinheitlichte Server-API



Apple Push Notification service (APN) Ports



TCP port 5223 (used by devices to communicate)

TCP port 2195 (used by devices to send notifications)

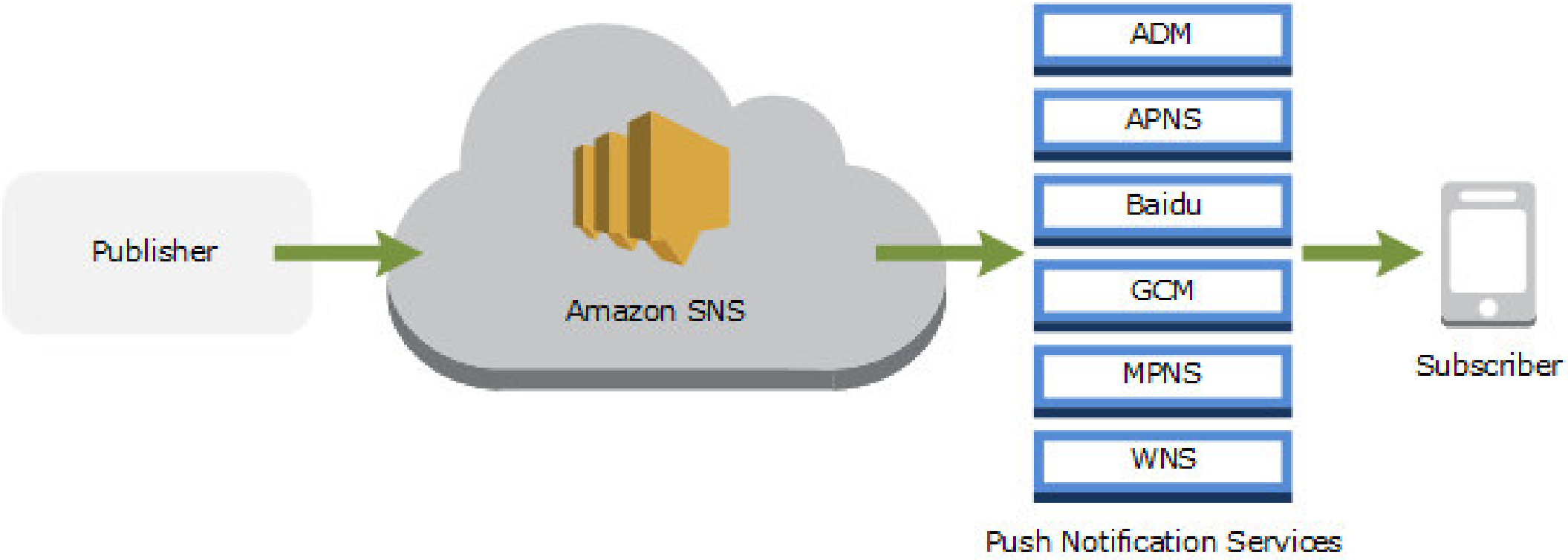
TCP port 2196 (used by the APNs feedback service)

TCP Port 443 (**Wi-fi fallback**, when unable to communicate on port 5223)

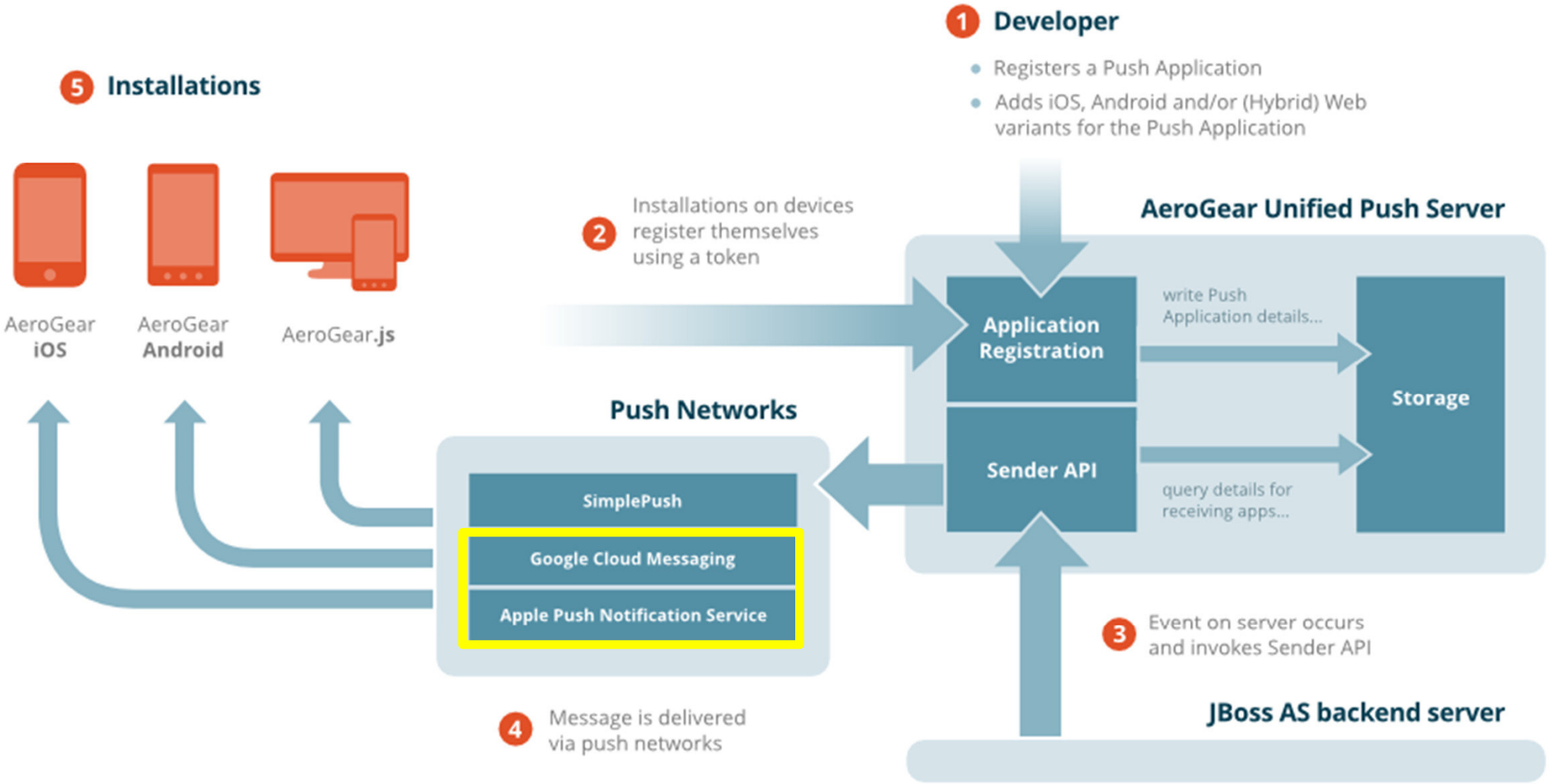
Google Cloud Messaging Kommunikationsarten
Cloud Connection Server (CCS)

- Upstream/Downstream messages
  - HTTP: Downstream only, cloud-to-device
  - XMPP: Upstream and downstream (device-to-cloud, cloud-to-device)
- Messaging (synchronous, asynchronous)
  - HTTP POST synchronous requests and wait for a response
  - XMPP: Asynchronous send/receive messages to/from devices at full over **persistent XMPP** connections. sends **acknowledgment** or failure notifications
- JSON
  - HTTP: JSON messages sent as HTTP POST
  - XMPP: JSON messages encapsulated in XMPP messages.
- Plain Text
  - HTTP: Plain Text messages sent as HTTP POST
- Multicast downstream send to multiple registration IDs
  - HTTP: Supported in JSON message format
- **Deletion** of already delivered messaging possible

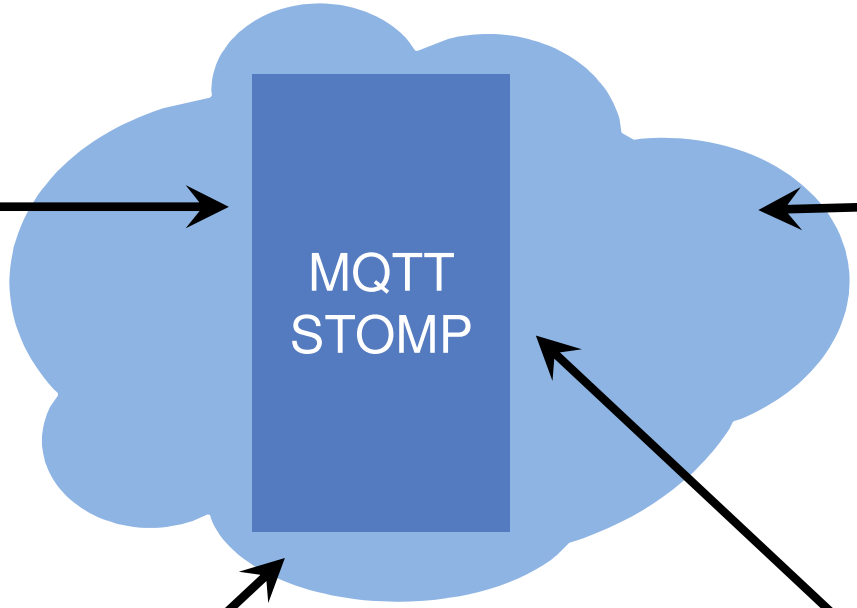
Lösung - Amazon SNS



Lösung - AeroGear Unified Push Server



Lösung - Mobile Nachrichten – vereinheitlichte Server-/Client-API/Protokoll



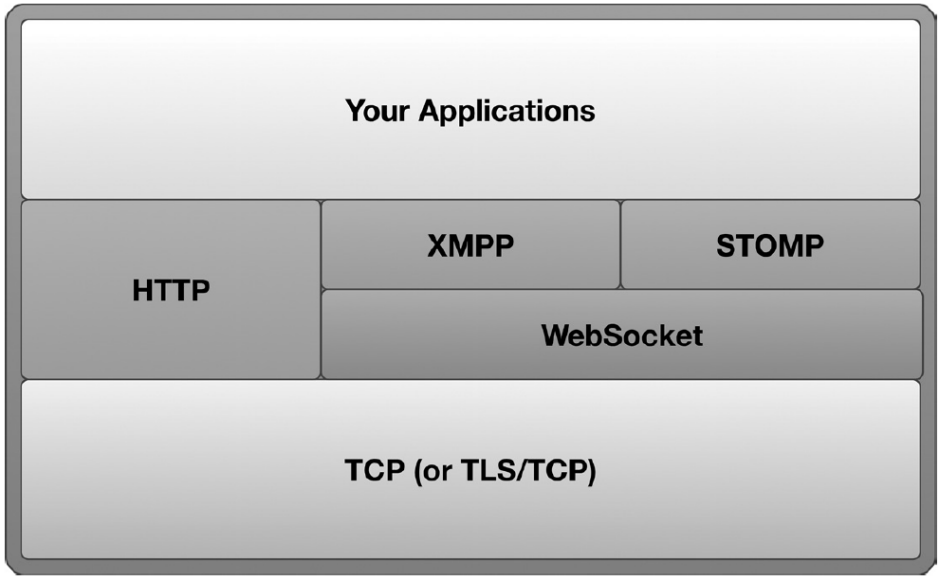
+Weniger Abhängigkeiten
+bessere Testbarkeit
+Standardisierung
-Server-Management
- Polyglotte Programmierung

Agenda

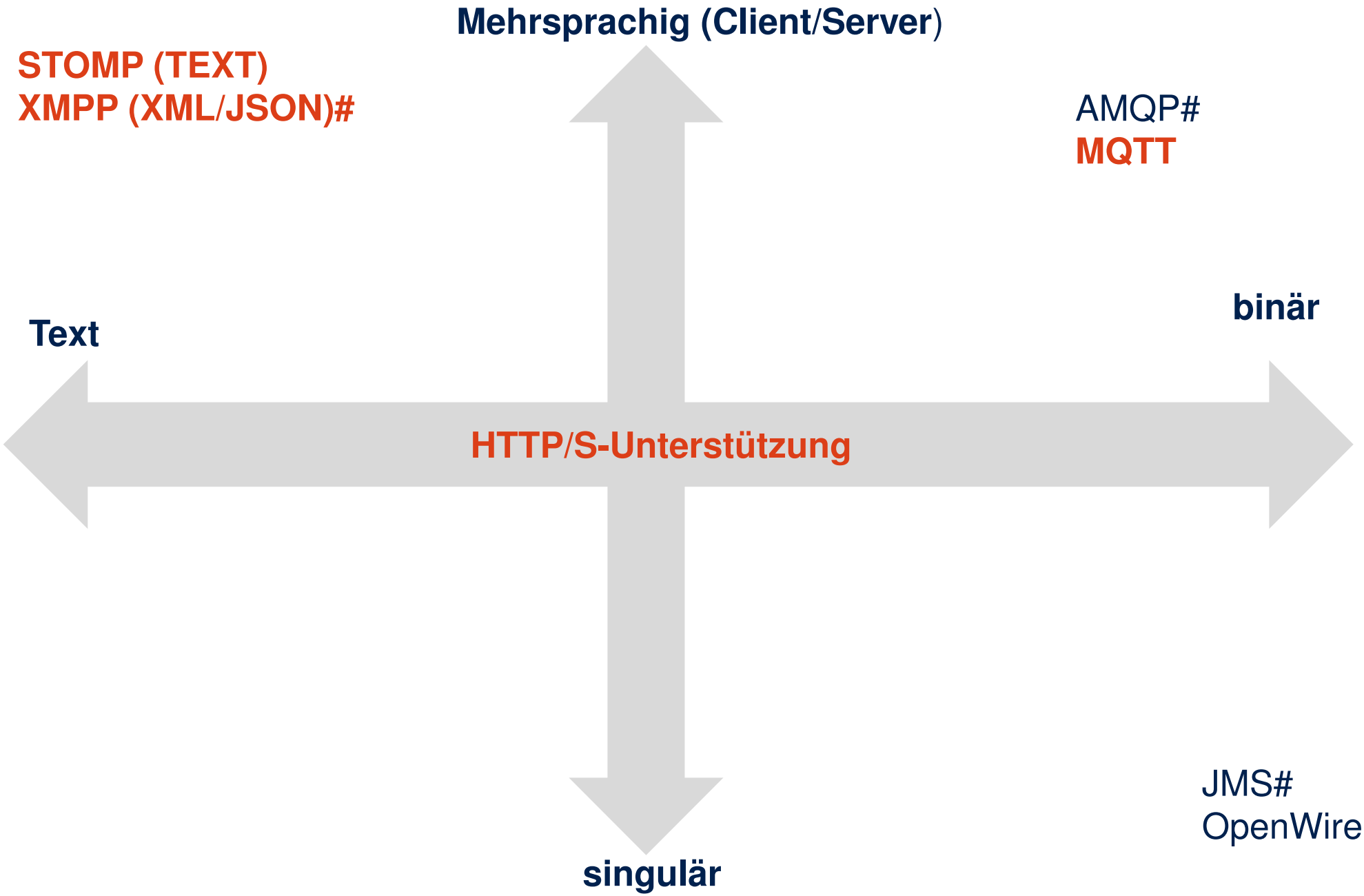
Kommunikation verändert sich	Mobiles Messaging – Möglichkeiten & Heraus-forderungen	Messaging-Protokolle – MQTT & STOMP	Messaging-Produkte für MQTT & STOMP	Beispiel (JavaScript, Java, CLI)	Fazit – wird möglich

Nachrichtenprotokolle

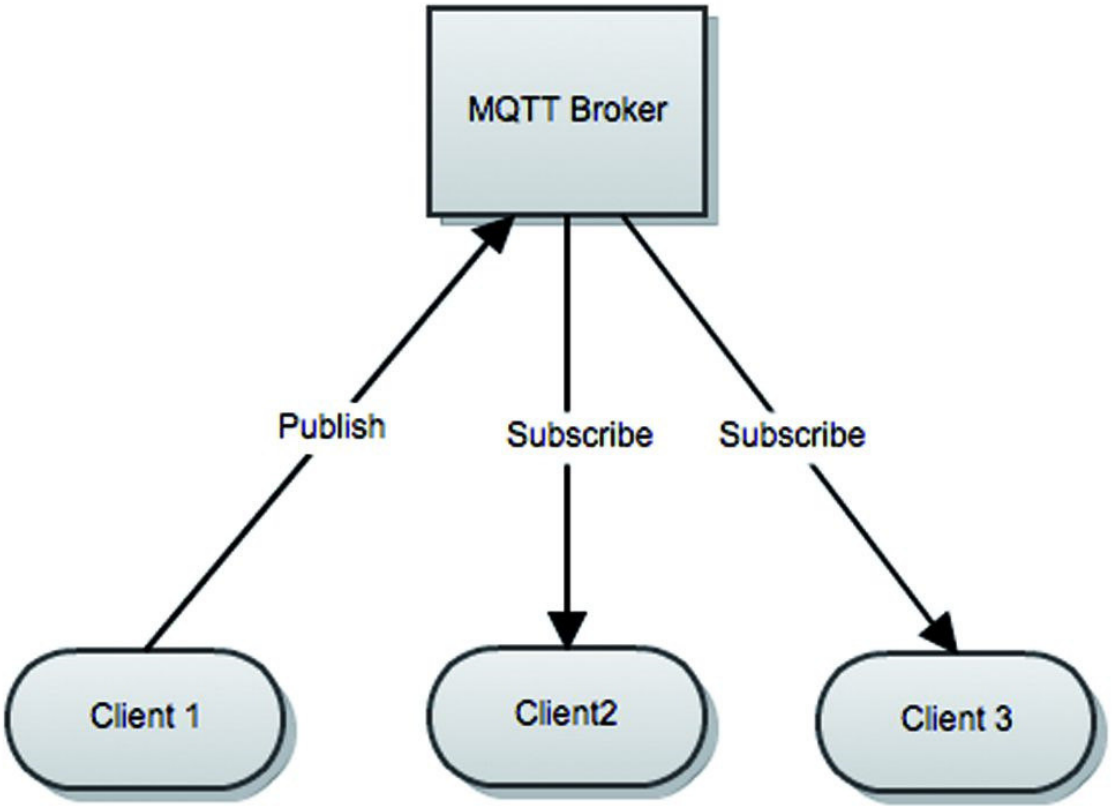
- **JMS** (Java Message Service) → JCP
- **AMQP** (Advanced Message Queuing Protocol) → OASIS, ISO/IEC 19464
- **OpenWire** → Apache License 2.0
- **STOMP** (Streaming Text Oriented Messaging Protocol) → (CC BY 3.0)
- **XMPP** (Jabber) → RFC/IETF
- **MQTT** (Message Queue Telemetry Transport) → OASIS



Nachrichtenprotokolle Einteilung

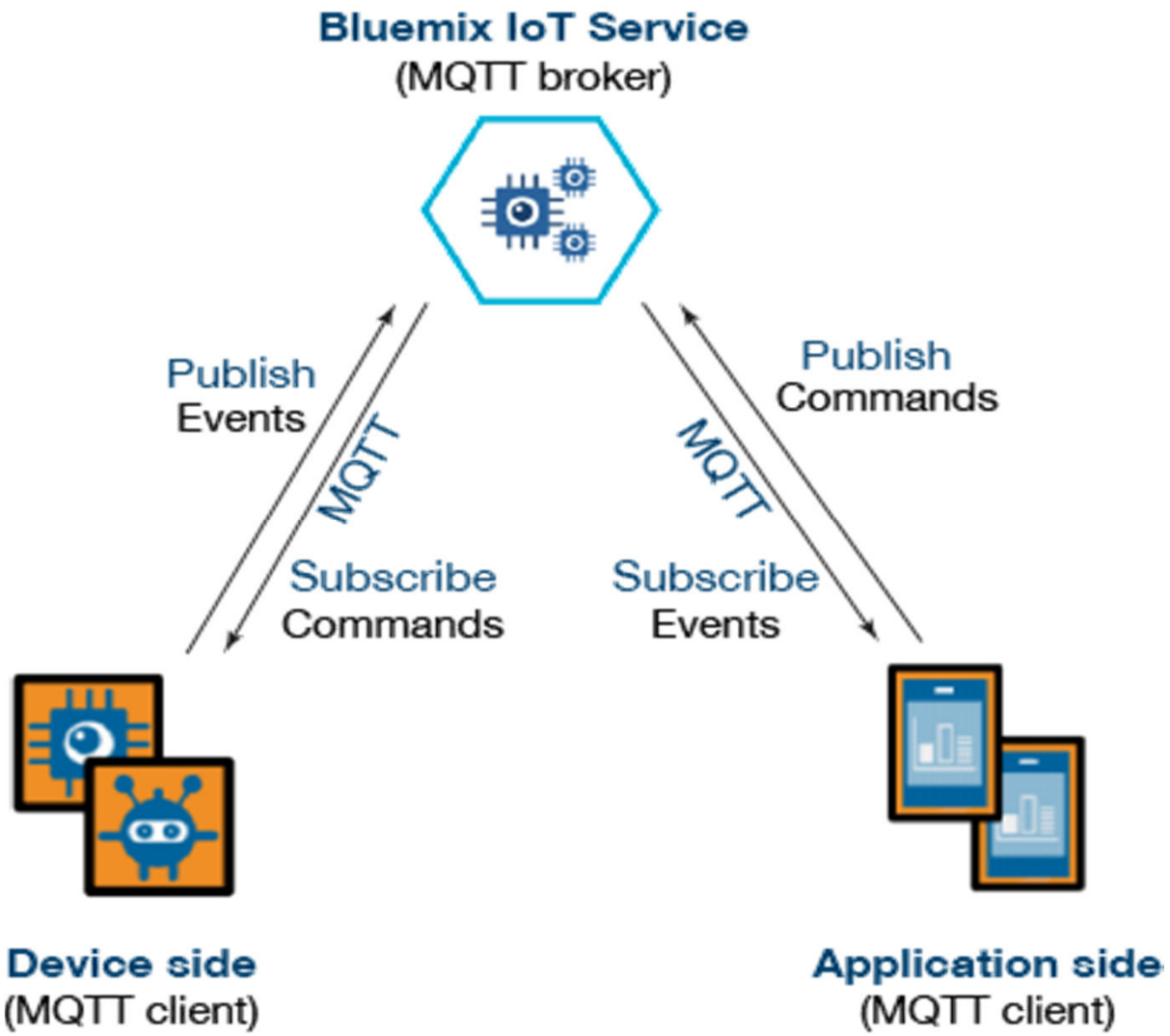


Message-Queue-Telemetry-Transport-(MQTT-)Protokoll

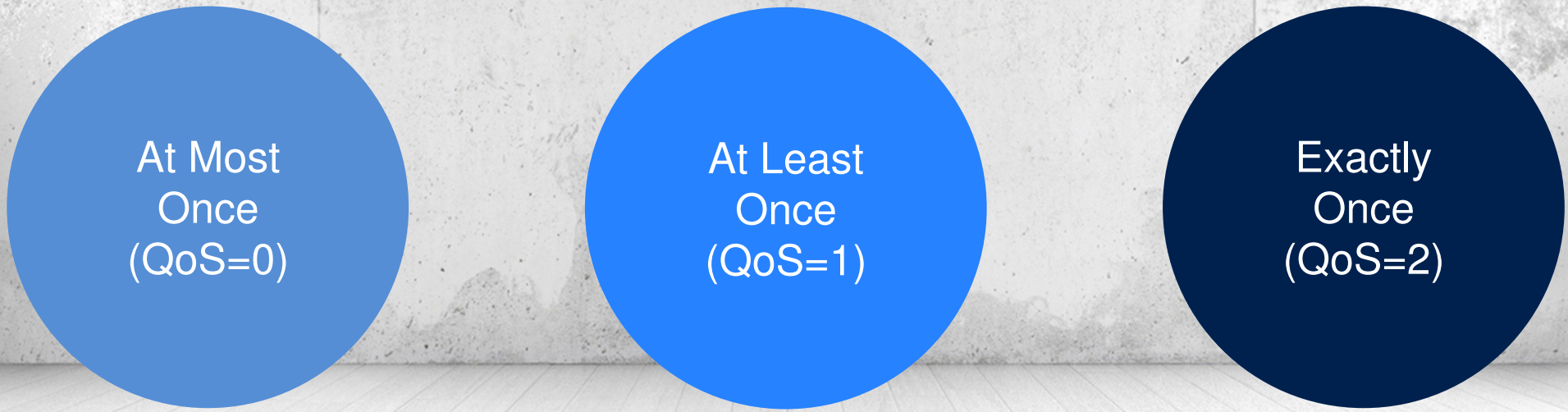


- **1999** von Andy Stanford-Clark (IBM) und Arlen Nipper (Eurotech)
- **2014** MQTT Protocol Specification Version 3.1.1

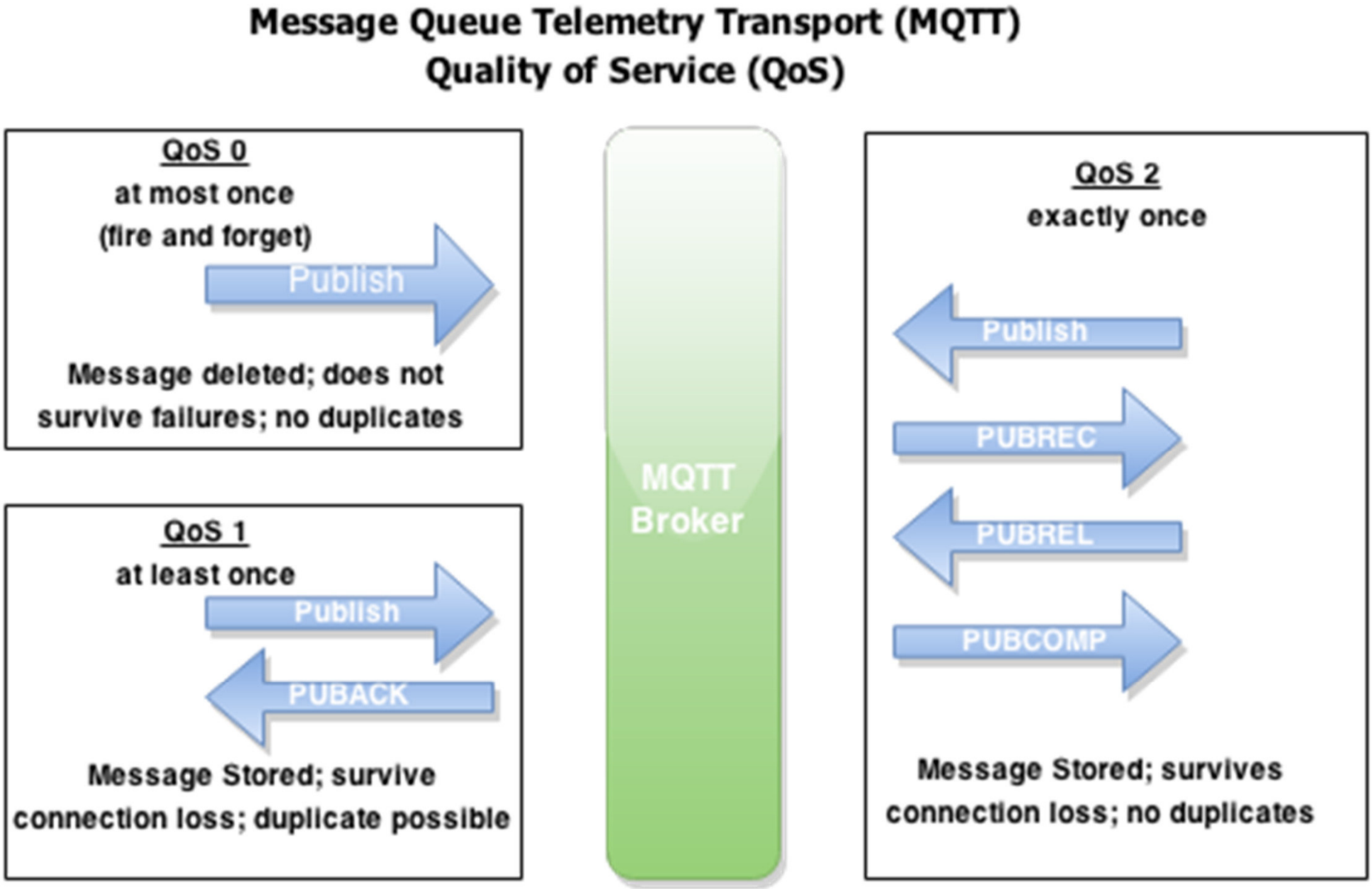
MQTT-Überblick Publish-Subscribe-Broker



MQTT Quality of Service (QoS)



MQTT Quality of Service Levels



Simple Text Orientated Messaging Protocol (STOMP) 1.2 (2012)

Einfaches, standardisiertes textbasiertes Protokoll zur freien Verwendung

```
client-command      = "SEND"
                    | "SUBSCRIBE"
                    | "UNSUBSCRIBE"
                    | "BEGIN"
                    | "COMMIT"
                    | "ABORT"
                    | "ACK"
                    | "NACK"
                    | "DISCONNECT"
                    | "CONNECT"
                    | "STOMP„
server-command      = "CONNECTED"
                    | "MESSAGE"
                    | "RECEIPT"
                    | "ERROR"
```

Agenda

Kommunikation verändert sich	Mobiles Messaging – Möglichkeiten & Heraus-forderungen	Messaging-Protokolle – MQTT & STOMP	Messaging-Produkte für MQTT & STOMP	Beispiel (JavaScript, Java, CLI)	Fazit – wird möglich

Eclipse Paho 1.1 und Eclipse Mosquitto 1.4

- **Paho-Projekt** http://www.eclipse.org/paho seit 2012 bei Eclipse
- **Ziel:** Implementierungen von Standard-Messaging-Client-Protokollen im Machine-to-Machine-Bereich (**M2M**) Clients für *Java, C, C++, JavaScript, Lua, Python, Go Client, C# .Net* und *WinRT*
- **Mosquitto-Server**: entstand 2010 als Open-Source-Implementierung des Internet-of-Things-Protokolls **MQTT 3.1.1**

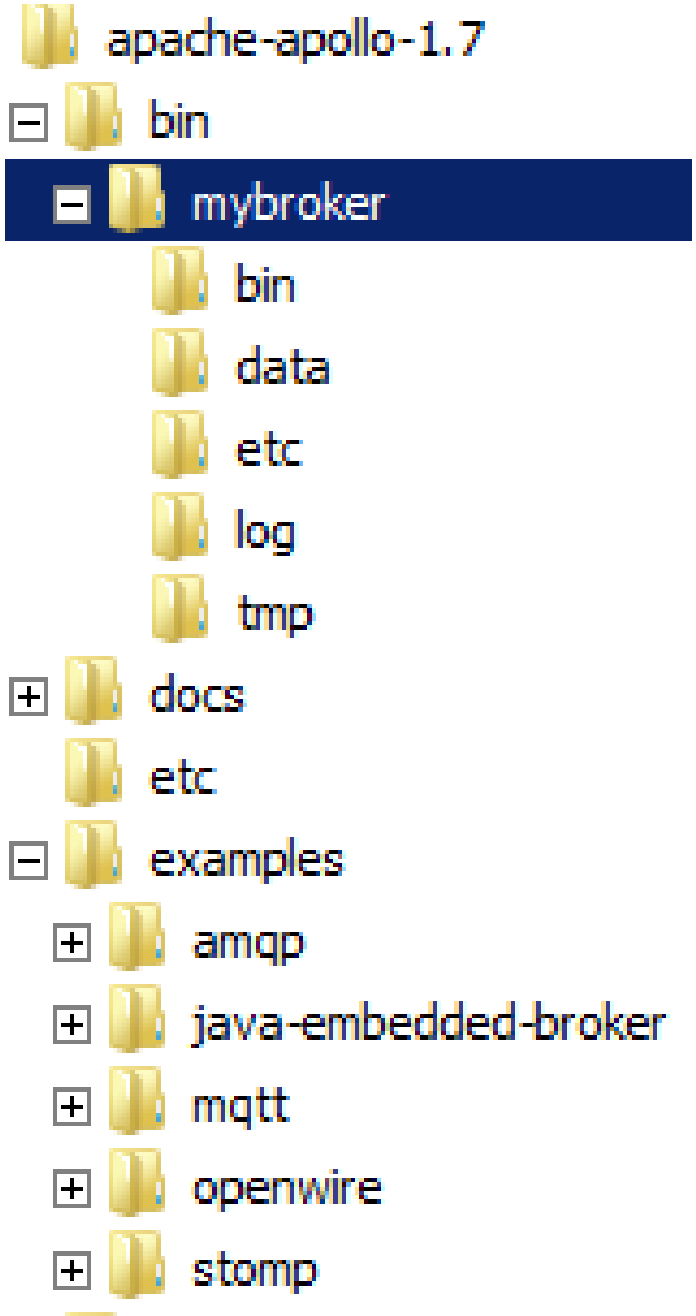
```
mosquitto.conf
mosquitto.dll
mosquitto.exe
mosquittopp.dll
mosquitto_passwd.exe
mosquitto_pub.exe
mosquitto_sub.exe

D:\mosquitto14>
```

Latest release: 1.1	Downloads
C client for Windows/Unix/Mac	1.0.3
Java client and utilities	1.0.2
Android service	1.0.2
Python client	1.1
JavaScript client	1.0.1
C/C++ MQTT Embedded clients	1.0.0
.Net and WinRT client (M2Mqtt)	4.0.0.0

Apache Apollo 1.7.1

- Apache **ActiveMQ Nachfolger**
- **Connectoren**
  - TCP
  - UDP
  - WebSocket
- **Protokolle**
  - **STOMP**
  - **MQTT**
  - OpenWire
  - AMQP

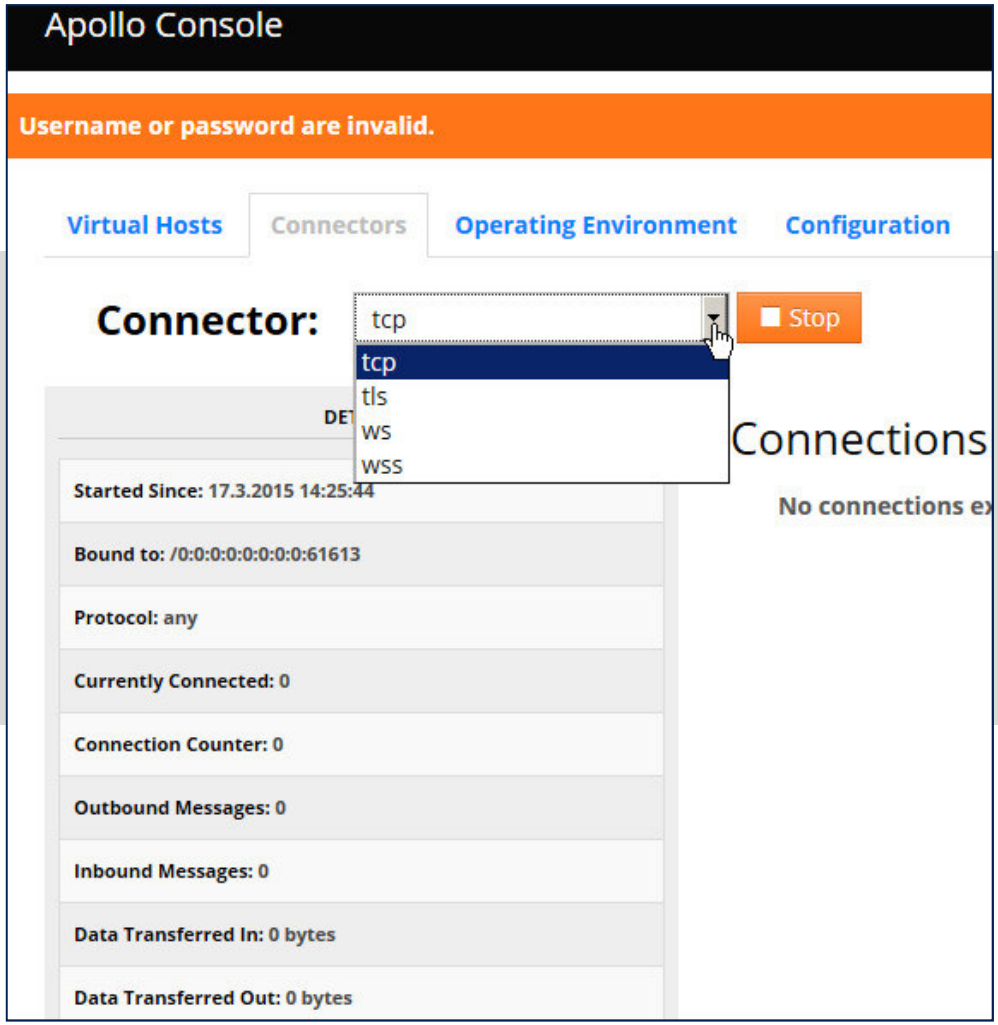


Apollo Broker apollo.xml

```
${APOLLO_HOME}/bin/apollo create mybroker
..\mybroker\bin\apollo-broker run
curl -u "admin:password" http://localhost:61680/broker
curl -u "admin:password" http://localhost:61680/broker.json
curl -u "admin:password" \
'http://localhost:61680/broker/connections.json'
```

```
Loading configuration file 'D:\apache-apollo-1.7\mybroker\etc\apollo.xml'.
INFO  | OS      : Windows 7 6.1
INFO  | JVM     : Java HotSpot(TM) 64-Bit Server VM 1.7.0_72 (Oracle Corporation)
INFO  | Apollo : 1.7 (at: D:\apache-apollo-1.7)
INFO  | Starting store: leveldb store at D:\apache-apollo-1.7\mybroker\data
INFO  | Accepting connections at: tcp://0.0.0.0:61613
INFO  | Accepting connections at: tls://0.0.0.0:61614
INFO  | Accepting connections at: ws://0.0.0.0:61623/
INFO  | Accepting connections at: wss://0.0.0.0:61624/
INFO  | virtual host startup is waiting on store startup
INFO  | virtual host startup is no longer waiting.  It waited a total of 1 seconds.
INFO  | Administration interface available at: https://127.0.0.1:61681/
INFO  | Administration interface available at: http://127.0.0.1:61680/
```

Apollo Webconsole



apollo.xml

- http://127.0.0.1:61680
- https://127.0.0.1:61681

users.properties

groups.properties

admin/password

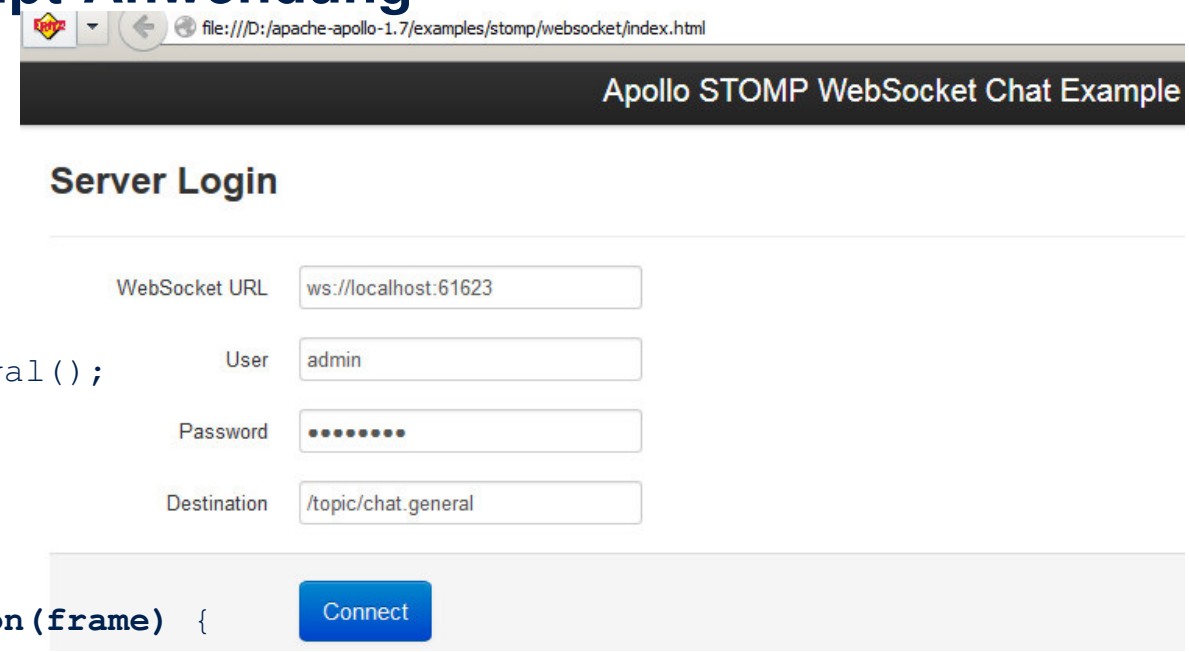
Agenda

Kommunikation verändert sich	Mobiles Messaging – Möglichkeiten & Heraus-forderungen	Messaging-Protokolle – MQTT & STOMP	Messaging-Produkte für MQTT & STOMP	Beispiel (JavaScript, Java, CLI)	Fazit – wird möglich

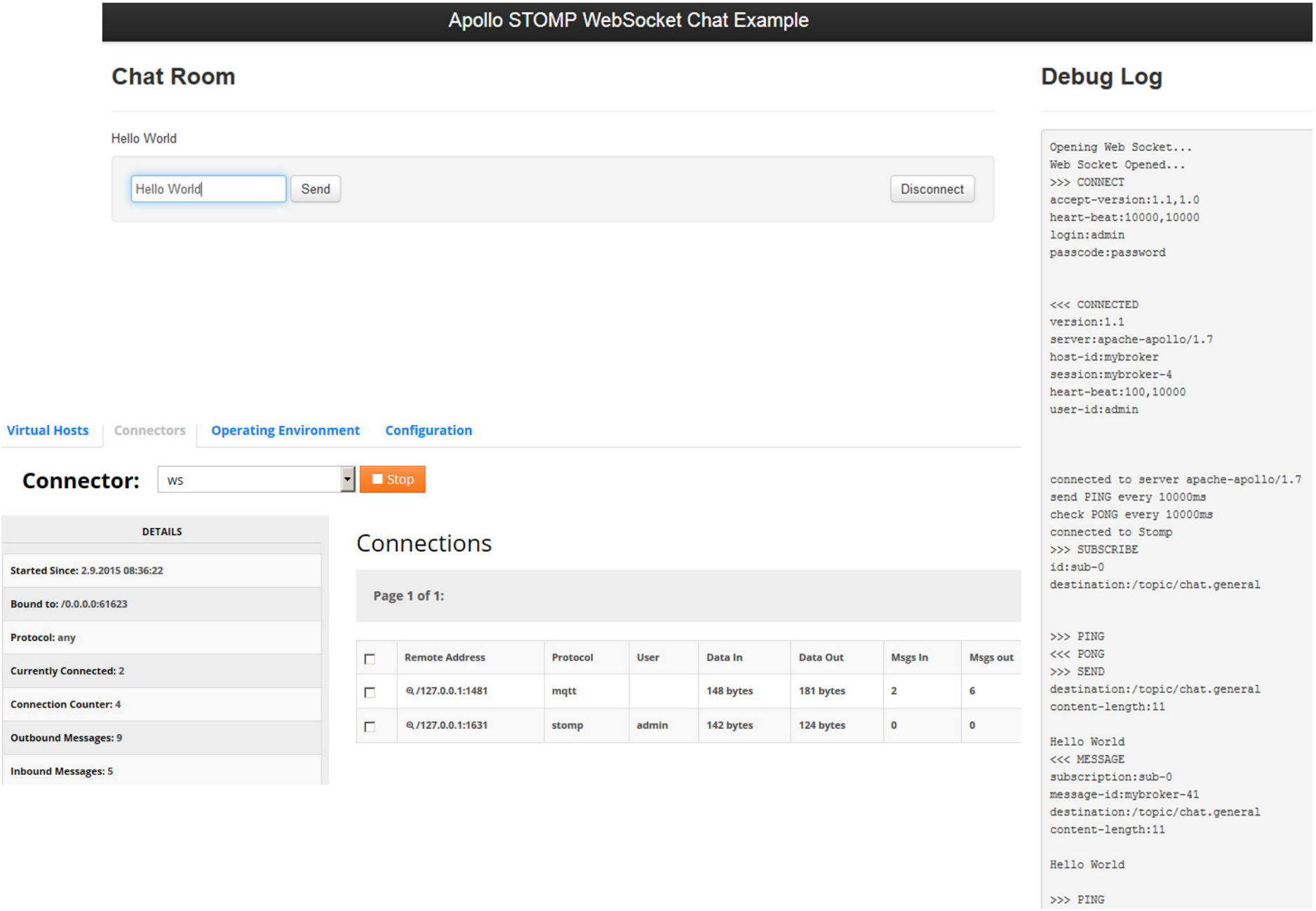
Apollo STOMP WebSocket JavaScript-Anwendung

```
<script src="js/stomp.js"></script>
    <script>//<![CDATA[
    $(document).ready(function() {
      if(window.WebSocket) {
        var client, destination;
        $('#connect_form').submit(function() {
          var url = $("#connect_url").val();
          var login = $("#connect_login").val();
          var passcode = $("#connect_passcode").val();
          destination = $("#destination").val();
          client = Stomp.client(url);
          client.debug = function(str) {
            $("#debug").append(str + "\n");
          };
          client.connect(login, passcode, function(frame) {
            client.debug("connected to Stomp");
            $('#connect').fadeOut({ duration: 'fast' });
            $('#connected').fadeIn();
            client.subscribe(destination, function(message) {
              $("#messages").append("<p>" + message.body + "</p>\n");
            });
          });
          return false;
        });

 $('#send_form').submit(function() {
    var text = $('#send_form_input').val();
    if (text) {
      client.send(destination, {}, text);
      $('#send_form_input').val("");
    }
 return false;         });
```



Apollo STOMP WebSocket Chat Anwendung



Apollo STOMP-Java-Anwendung

```
String user = env("APOLLO_USER", "admin");
String password = env("APOLLO_PASSWORD", "password");
String host = env("APOLLO_HOST", "localhost");
int port = Integer.parseInt(env("APOLLO_PORT", "61613"));
String destination = arg(args, 0, "/topic/event");

StompJmsConnectionFactory factory = new StompJmsConnectionFactory();
factory.setBrokerURI("tcp://" + host + ":" + port);

Connection connection = factory.createConnection(user, password);
connection.start();
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
Destination dest = new StompJmsDestination(destination);
MessageProducer producer = session.createProducer(dest);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

for( int i=1; i <= messages; i ++) {
    TextMessage msg = session.createTextMessage(body);
    msg.setIntProperty("id", i);
    producer.send(msg);
    if( (i % 1000) == 0) {
        System.out.println(String.format("Sent %d messages", i));
    }
}
producer.send(session.createTextMessage("SHUTDOWN"));
connection.close();
```

CLI-CRUD mit MQTT-Mosquitto-Server tcp://iot.eclipse.org:1883

- Über **HTTP-to-MQTT Bridge**

```
curl -X PUT --data-binary "Hello World"
http://eclipse.mqttbridge.com/JS curl
http://eclipse.mqttbridge.com/JS
curl -X DELETE http://eclipse.mqttbridge.com/JS
```

- Mit Mosquitto-Client Thema JS abonnieren

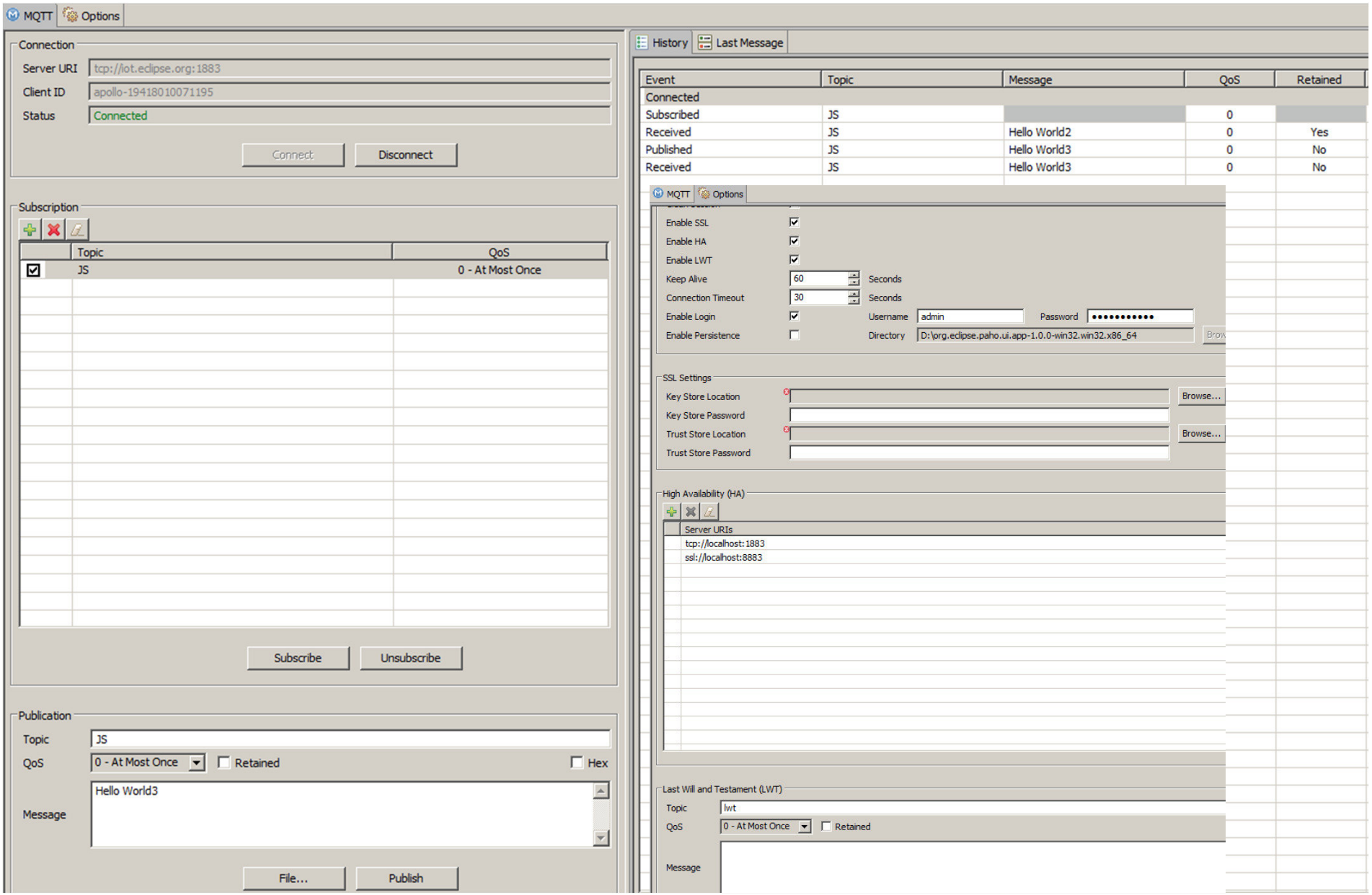
```
mosquitto_sub -h iot.eclipse.org -p 1883 -t JS -v
```

- Nachrichten an ein bestimmtes Thema senden

```
mosquitto_pub -h iot.eclipse.org -p 1883 -r -t JS -m
"Hello World"
```

- Etwas komfortabler geht das mit der Eclipse RCP-Anwendung MQTT Client Tools

Eclipse Paho MQTT Client und MQTT-Mosquitto-Server tcp://iot.eclipse.org:1883



Eclipse Paho MQTT Java-Client

```
String topic        = "JS";
String content      = "Hello World";
int qos             = 0;
String broker       = "tcp://iot.eclipse.org:1883";
String clientId     = "JSSample";
MemoryPersistence persistence = new MemoryPersistence();
MqttClient sampleClient = new
MqttClient(broker,clientId,persistence);
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setCleanSession(true);
System.out.println("Connecting to broker: " + broker);
sampleClient.connect(connOpts);
System.out.println("Connected");
System.out.println("Publishing message: " + content);
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(qos);
sampleClient.publish(topic, message);
System.out.println("Message ” + content + “  published");
sampleClient.disconnect();
```

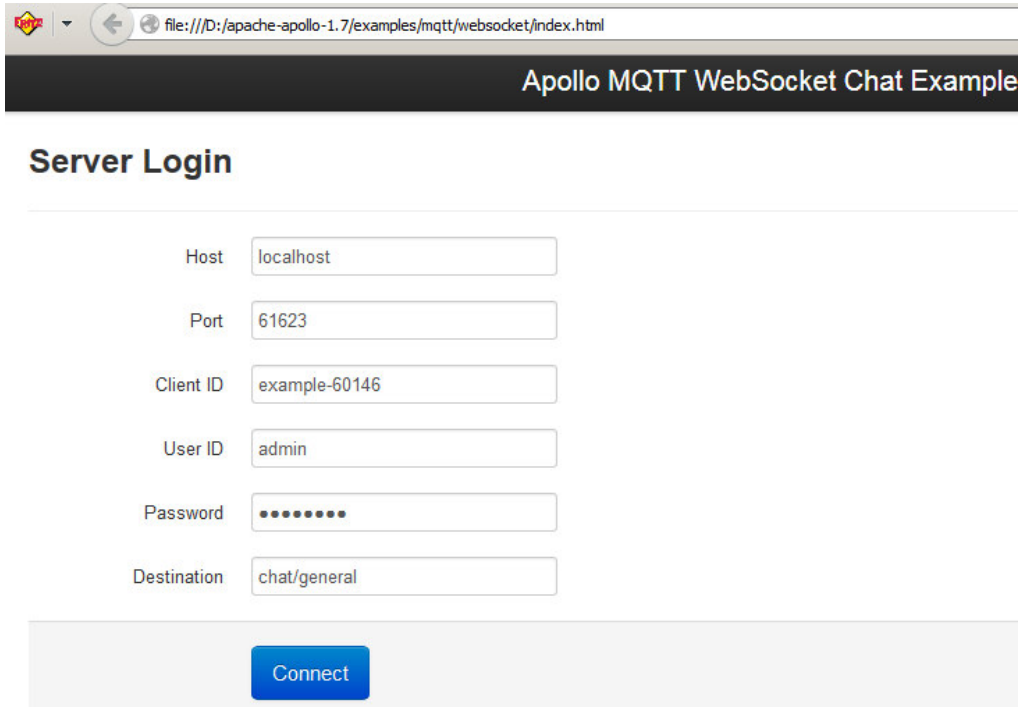
Apollo MQTT-Java-/JavaScript-Anwendung

apache-apollo-1.7\examples\mqtt\java\src\main\java\example

- `Listener.java`
- `Publisher.java`

websocket\index.html

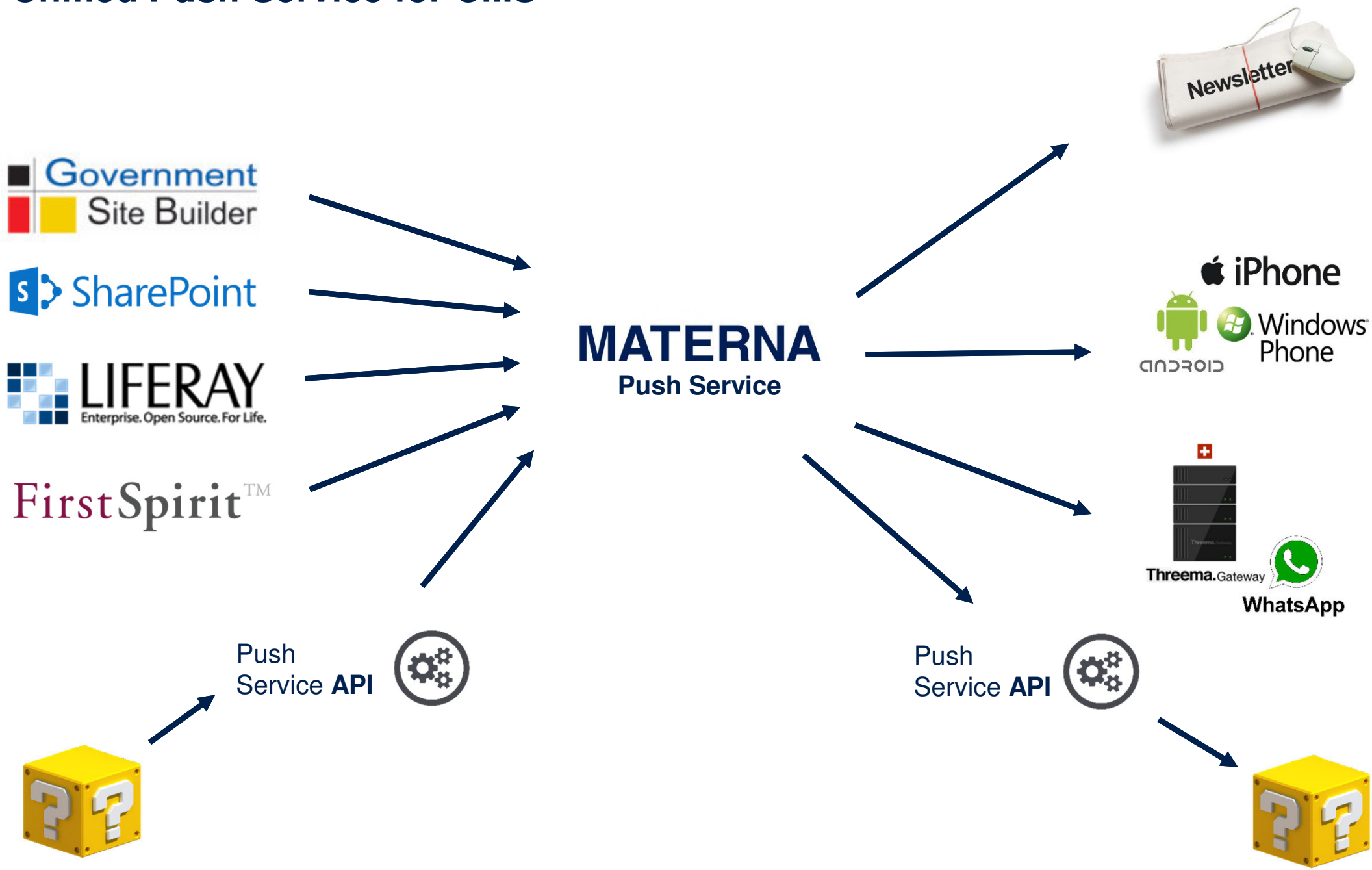
- `js/mqttws31.js`



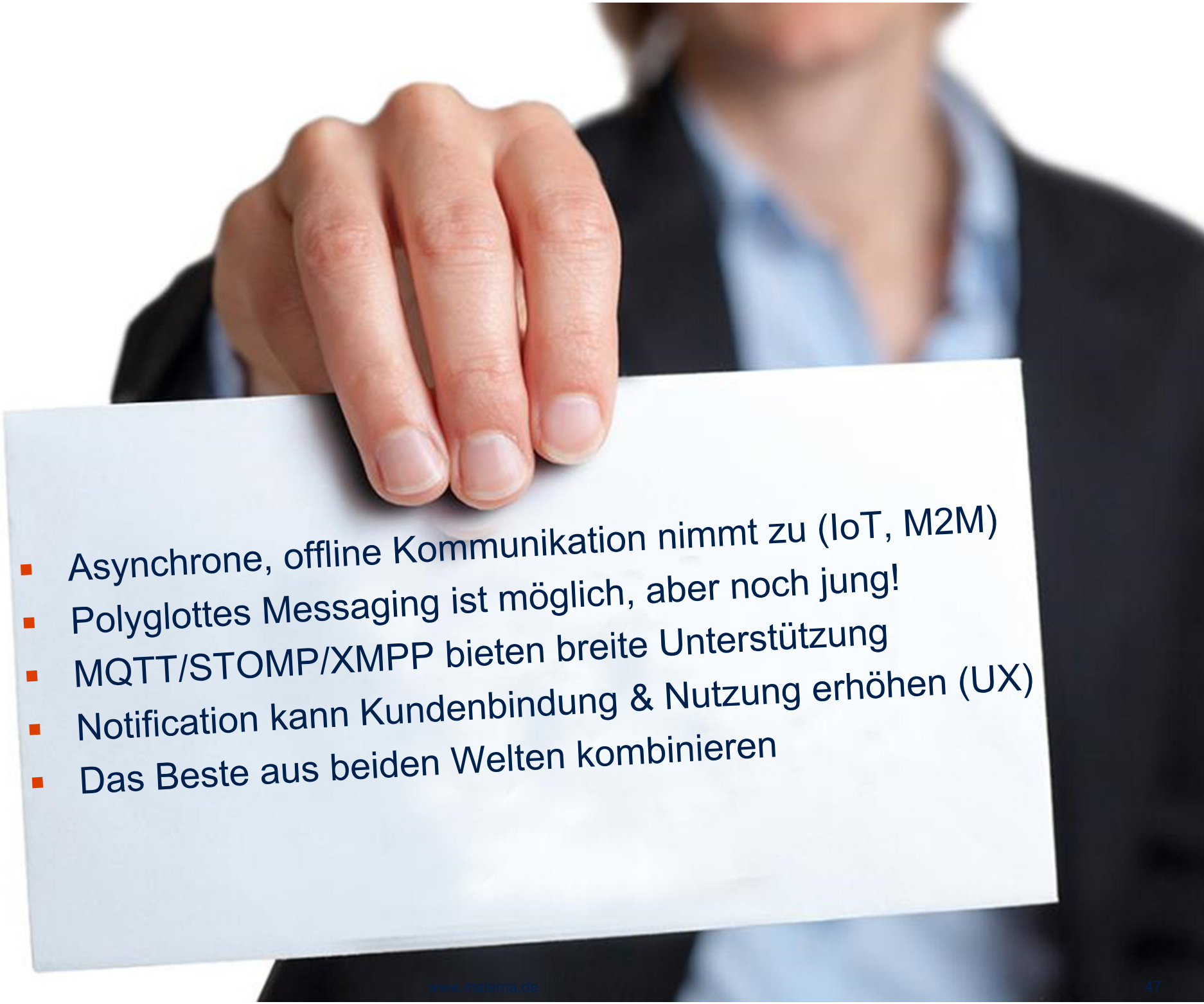
Agenda

Kommunikation verändert sich	Mobiles Messaging – Möglichkeiten & Heraus-forderungen	Messaging-Protokolle – MQTT & STOMP	Messaging-Produkte für MQTT & STOMP	Beispiel (JavaScript, Java, CLI)	Fazit – wird möglich

Unified Push Service for CMS

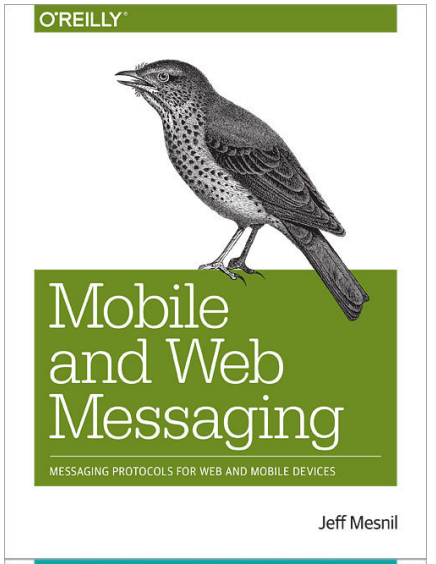


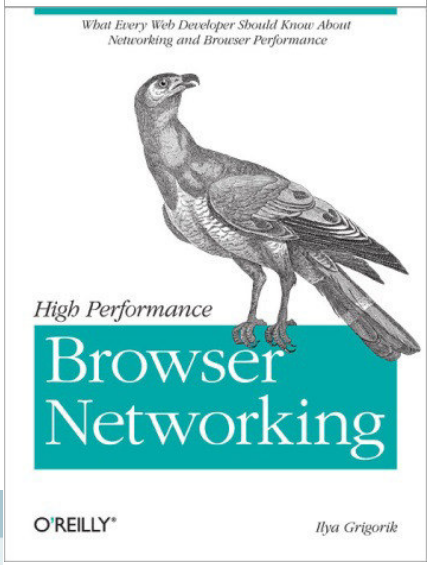
Fazit

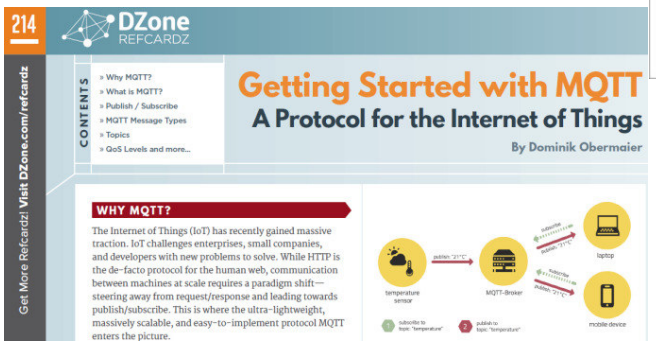


Links

- STOMP Protokoll-Spezifikationen http://stomp.github.io
- MQTT-Organisation http://mqtt.org
- MQTT-Spec http://docs.oasis-open.org/mqtt/mqtt/v3.1.1
- MQTT.js https://www.npmjs.com/package/mqtt
- Public MQTT Brokers https://github.com/mqtt/mqtt.github.io/wiki/public_brokers
- Eclipse Paho http://eclipse.org/paho
- Mosquitto-Server http://mosquitto.org
- Apache Apollo https://activemq.apache.org/apollo
- Google Cloud Messaging (GCM) Server https://developer.android.com/google/gcm/server.html
- https://dzone.com/refcardz/getting-started-with-mqtt
- AeroGear http://aerogear.org








Fragen?

MATERNA
Information & Communications

Vernetzt.

Kontakt.

Materna GmbH
Frank Pientka
Voßkuhle 37
44141 Dortmund
+49 231 5599-8854
Frank.Pientka@materna.de